高等学校计算机应用实践教程系列

ASP.NET 3.5 应用实践教程

郑阿奇　主编
王志瑞　编著

電子工業出版社
Publishing House of Electronics Industry
北京・BEIJING

内容简介

本书以 ASP.NET 35 应用开发实例驱动，把多且复杂的知识点分散开来，通过应用过程，系统介绍 ASP.NET 3.5 的基本技术和方法。本书既可以用于课堂教学，又可以在计算机教室或机房演示教学，是软件开发教材的新尝试。本书结构和内容如下：应用系统概览、创建系统网站项目、设计数据库结构与构建访问层、创建学生成绩管理系统主框架、学生信息注册与系统登录、系统基础数据维护模块、设计成绩录入与打印模块、系统发布与部署，以及相关的附录内容。本书免费提供教学课件和源代码。

本书可以作为大学本科和高职高专 ASP.NET 3.5 应用开发课程或课程实习教材，也非常适合 ASP.NET 3.5 应用开发技术培训和应用开发参考。

图书在版编目（CIP）数据

ASP.NET 3.5 应用实践教程 / 郑阿奇主编；王志瑞编著. —北京：电子工业出版社，2010.3
（高等学校计算机应用实践系列教材）
ISBN 978-7-121-10535-7

Ⅰ. A… Ⅱ. ①郑… ②王… Ⅲ. 主页制作—程序设计—高等学校—教材 Ⅳ. TP393.092

中国版本图书馆 CIP 数据核字（2010）第 043842 号

策划编辑：童占梅
责任编辑：史鹏举
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编　100036
开　　本：787×1 092　1/16　印张：15.75　字数：403 千字
印　　次：2010 年 3 月第 1 次印刷
印　　数：4 000 册　　定价：25.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

ASP.NET 3.5 是目前开发 B/S 应用系统的最佳平台，我国高校的许多专业开设 ASP.NET 应用程序设计课程。我们根据教学和应用开发的经验，编写了《ASP.NET 3.5 应用实践教程》，比较系统地介绍了 ASP.NET 3.5 及其应用开发技术。

现有介绍 ASP.NET 开发的图书一般属于指南性质，书很厚，不适合教学。介绍 ASP.NET 内容的教材主要包括两种方式。一是传统的方式。这种方式按部就班，虽然比较系统，但训练动手能力的效果欠佳。二是实例方式。这种方式应用性较好，主要是模仿学习，但缺乏系统性，其训练解决新问题的能力和效果欠佳。

通过一段时间的思考，我们提出把上述两种方式的优点结合起来，尽可能补充各自的不足，应用实践教程系列就想在这方面进行探索。

《ASP.NET 3.5 应用实践教程》以当前最先进的 Microsoft Studio 2008 作为平台，介绍 ASP.NET 3.5 应用开发技术。编写思路如下：

（1）以一个应用系统作为引导，同时跟着完成这个应用系统。由于应用系统结构和包含的内容精心设计，能够确保学生在此过程中基本掌握 ASP.NET 3.5 主要内容。

（2）在介绍功能的同时通过超链接关联知识点，让学生不但知道怎么做，而且知道为什么这么做。超链接关联的方法采用与章节同名的方式，另加“L”加以区分，查找简单方便。如右图所示。

为了达到本书的教学目标，教材编写人员经过多次修改和完善，然后由有关人员跟着本书进行试做审读，发现问题后由教材编写人员继续修改和完善，在这个前提下交下一环节。当然，我们并不认为本书有多完美，只是认为这是一种软件开发教材的新尝试，应

第一部分　应 用 实 践

2.1　创建系统网站项目

创建系统网站项目步骤如下：

（1）运行 Visual Studio 2008，默认显示出“起始页”，在“起始页”内可以查看微软官方 MSDN 最新的技术文档和最近开发的项目列表。

（2）选择“文件”→“新建”→“网站”菜单项，打开“新建网站”窗口，如图 2.2 所示。

（3）在“新建网站”窗口已安装的模板中选择“**ASP.NET 网站**”模板，在窗口的顶部下拉框内，选择“**.NET Framework 3.5**”框架，在“语言”下拉框中选择“Visual C#”。

（4）单击“浏览”按钮，出现如图 2.3 所示的“选择位置”对话框，有 4 个站点位置选项，默认选项是“文件系统”，“文件系统”方式允许把所要创建的站点文件保存到电脑磁盘的任何位置。选中 D 盘，单击 按钮，在 D 盘下创建一个“学生成绩管理系统”文件夹。选中此文件夹，单击“打开”按钮，返回到“新建网站”窗口，此时“新建网站”窗口的“位置”行中的下拉列表框会选中“文件系统”，对应的路径为“D:\学生成绩管理系统”，单击“确定”按钮，完成站点的创建工作。

……

第二部分　知识点链接

L2.1　创建系统网站项目

L1.　ASP.NET 网站

Visual Studio 2008 是.NET 环境下功能比较完备的开发平台，可以用来开发很多种项目，为不同的项目提供了不同的项目模板。需要建立网站项目，可以通过“文件”→“新建”→“项目”菜单项，打开“新建项目”窗口，如图 2.6 所示，在此窗口左侧选择“Web”，右侧会出现 ASP.NET 支持的项目类型，选择其中的“ASP.NET Web 应用程序”。同样也可以通过“文件”→“新建”→“网站”菜单项，快速地创建 ASP.NET 网站项目。

该有耳目一新的感觉。采用本书学习 ASP.NET 3.5，教师和学生更轻松些，而且一定能够做出一些东西，一定能够学到一些东西。

本书由南京师范大学郑阿奇主编，王志瑞编著，参加本套丛书编写的还有郑进、陶卫冬、邓拼搏、严大牛、卢海艇、韩翠青、王海娇、刘博宇、丁有和、陈瀚、周怡明、吴明祥、刘毅、孙德荣。

许多同志参加了本书的基础工作，在此一并表示感谢！

本书配有教学课件和应用系统的所有源代码文件，需要者可以在华信教育资源网免费注册下载。网站地址为 http://www.hxedu.com.cn。

由于我们的水平有限，错误在所难免，敬请广大师生、读者批评指正。

意见建议邮箱：easybooks@163.com。

编　者

目　录

第 1 章　应用系统概览

本章主要介绍教材的主要目标和内容安排，包括系统所要实现的功能及 ASP.NET 知识的分布情况，最后给出系统的功能实现预览和系统结构流程图。

1.1　应用系统介绍

学生成绩管理系统是学校内部必不可少的学生信息管理系统，也是同学们经常接触到的管理系统。因此采用本系统作为例子，对于学生而言能够快速地理解系统需求，并对系统内部的基本逻辑比较好掌握。

本书的应用实践部分主要以实现学生成绩管理系统为例，讲解利用 ASP.NET 开发信息管理系统的基本方法与方式。教材内容按照开发信息管理系统的基本步骤为主要线索，并根据 ASP.NET 的知识体系，选取实现了系统内的相关功能模块。通过本教材既可以系统地掌握 ASP.NET 知识体系，又能够掌握如何利用 ASP.NET 进行实际系统开发，改变了以往教材只讲知识点，无法与实际应用相结合的弊端，通过这种方式既能有效地提高学生学习的积极性，又能提高学生的学习效果。

1.2　系统需求分析

本节将针对学生成绩管理系统的总体功能需求进行分析。这里介绍的学生成绩管理系统的需求分析只是一个精简的版本，在实际应用中应根据客户的不同需求，在此基础上进行扩展。

1.2.1　确定系统参与者

在需求分析阶段，首先需要确定系统的参与者。确定参与者首先需要分析系统所涉及的问题领域和系统运行的主要任务：分析使用该系统的是哪些人，谁需要该系统的支持以完成其工作，以及确定系统的管理者和维护者。

根据学生成绩管理系统的需求分析，可以确定如下几点：

（1）作为学生成绩管理系统，首先需要有学生的参与，学生可以登录系统查看课程的安排情况、查看课程成绩信息等。

（2）对于此学生成绩管理系统，系统的维护也是相当重要的。维护操作主要包括教工的增改删查、课程的增改删查、班级的增改删查及学生的增改删查等，因此需要系统管理员的参与，在此学生成绩管理系统中系统管理员主要是教务人员。

（3）作为学生成绩管理系统，当然要有教师的参与，教师要有录入学生考试成绩功

能，同时也要有打印分析学生成绩功能等。

由以上分析可以得出，系统的参与者主要有 3 类：学生、教师、教务人员。

1.2.2 确定系统用例

用例是系统参与者与系统在交互过程中所需要完成的事务，是从使用者的角度分析系统应该具有的功能。识别用例最好的办法就是从分析系统的参与者开始，考虑每个参与者如何使用系统的。由于学生成绩管理系统存在学生、教师、教务人员 3 种参与者，所以在识别用例的过程中，可以分别考虑不同的参与者。

1．学生使用本系统的相关用例

（1）登录/退出系统；
（2）查询课程安排情况；
（3）向老师提供课程问题；
（4）查询课程的考试成绩。

2．教务人员（管理员）使用本系统的相关用例

（1）登录/退出系统；
（2）增加、删除、修改、查询教工信息；
（3）增加、删除、修改、查询课程信息；
（4）增加、删除、修改、查询班级信息；
（5）增加、删除、修改、查询学生信息；
（6）增加、删除、修改、查询课程安排信息；
（7）查询学生成绩；
（8）分析学生成绩；
（9）打印学生的成绩单。

3．教师使用本系统的相关用例

（1）登录/退出系统；
（2）给学生解答课程问题；
（3）录入学生成绩；
（4）查询学生成绩；
（5）分析学生成绩；
（6）打印学生成绩单；
（7）退出系统。

1.3 系统功能与知识点对应章节

1．本书主要实现的系统功能

（1）系统网站的建立；
（2）系统主框架的构建；

（3）学生信息注册；
（4）学生成绩信息查询；
（5）系统登录；
（6）课程信息维护；
（7）教师信息维护；
（8）课程安排信息维护；
（9）成绩录入；
（10）成绩查询与打印；
（11）系统退出；
（12）站点发布与部署。

2．本书介绍的ASP.NET知识点

（1）系统项目创建；
（2）主题与母版页；
（3）HTML 与 CSS；
（4）基本服务器控件；
（5）内置对象；
（6）数据源控件；
（7）数据绑定控件；
（8）水晶报表；
（9）站点发布。

3．知识点与功能的章节分布

知识点与功能的章节分布情况如下。

第 2 章：介绍如何创建系统网站项目，以及 Visual Studio 2008 的基本开发环境。

第 3 章：介绍本系统的数据库分析与设计方法，同时介绍了如何对 ADO.NET 进行封装以提高后期开发的效率。

第 4 章：介绍利用母版页构建系统内子系统的主界面，并介绍了如何利用主题与 CSS 样式控制页面的布局与展现。

第 5 章：介绍利用基本服务器控件创建学生注册页面及用户登录页面。

第 6 章：介绍利用数据源控件与数据绑定控件快速地实现数据表的查询、修改、添加、删除等基本的数据库操作。

第 7 章：介绍如何动态地控制数据源与数据绑定控件实现一些复杂的功能，并介绍如何利用水晶报表实现数据打印、学生成绩查询和退出系统功能。

第 8 章：介绍如何生成与发布站点、如何把系统部署到实际的运行环境中。

1.4　系统功能预览

1．网站项目创建及构建基本的文件结构

网站项目创建及构建基本的文件结构，如图 1.1 所示。

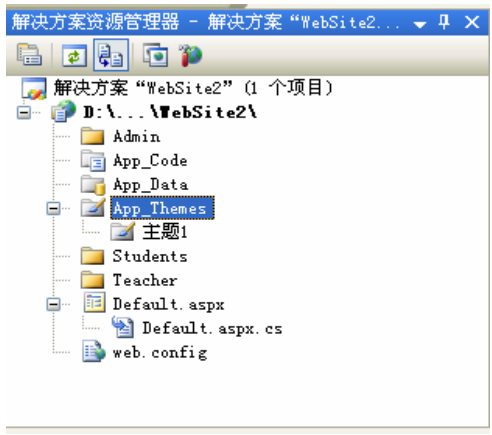


图 1.1　网站目录结构

其中，学生、教师、教务人员 3 种参与者需要系统对应 3 种不同的操作功能。系统创建 Students、Teacher 和 Admin 文件夹，分别存放对应 3 种参与者的系统实现文件。其他的文件夹和文件将在后面介绍。

2．管理员子系统与教师子系统的主框架

管理员子系统需要实现的功能如图 1.2 所示。

图 1.2　管理员子系统主框架

教师子系统需要实现的功能如图 1.3 所示。

图 1.3　教师子系统主框架

3．学生信息注册

学生在使用系统前必须进行注册，注册完成后进行系统登录进入系统。系统注册界面如图 1.4、图 1.5 所示。

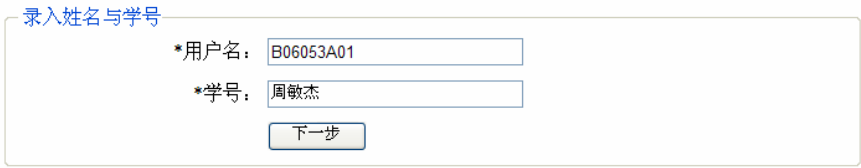


图 1.4　用户注册页面（1）

图 1.5　用户注册页面（2）

4．系统登录页面

所有人员必须通过登录才能进入系统。系统登录页面如图 1.6 所示。

图 1.6　系统登录页面

系统登录时，首先选择用户类别，包括学生、教师、教务人员，然后单击“登录”按钮进入系统。前面已经说过，学生使用系统前需要单击“学生注册”按钮进行注册。

5．系统基本信息的维护

系统基本信息的维护包括课程信息维护、教师信息维护、排课信息维护等。课程信息维护页面如图 1.7 所示。

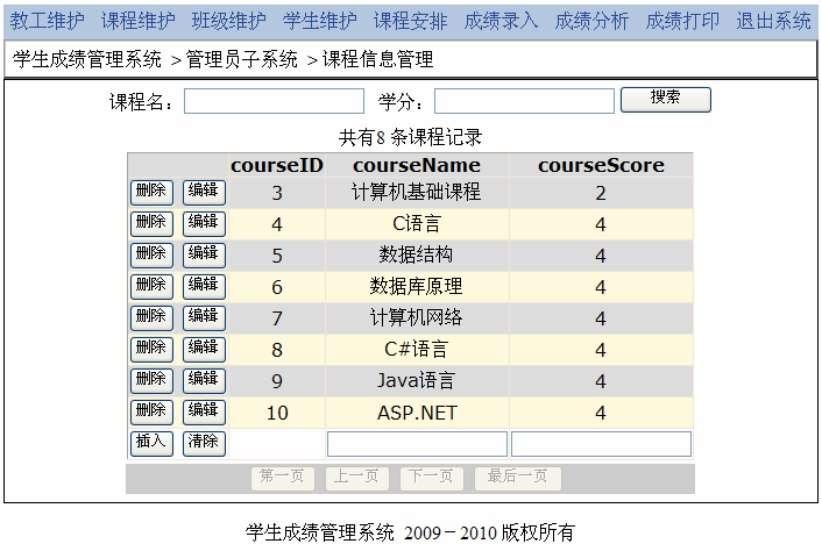


图 1.7　课程信息维护页面

当前已经输入的课程信息（课程名和学分）以表格形式显示，并且通过页面下部的“第一页”、“上一页”、“下一页”和“最后一页”按钮进行分页显示。表格上部显示已经输入的课程名，通过页面上部的“搜索”按钮可以查找是否存在指定课程。新的课程信息在表格的下部输入后“插入”数据库表。

教师信息维护页面如图 1.8 所示。

图 1.8　教师信息维护页面

其中，页面的中间以表格显示已经输入的教师信息，并可以进行修改、删除和显示详情。表格的上部包含对已经输入的教师数进行计数，教师详情信息在表格下部输入、修改、显示等。页面的上部可以按“姓名”和“手机号码”进行查找。

排课信息维护页面如图 1.9 所示。

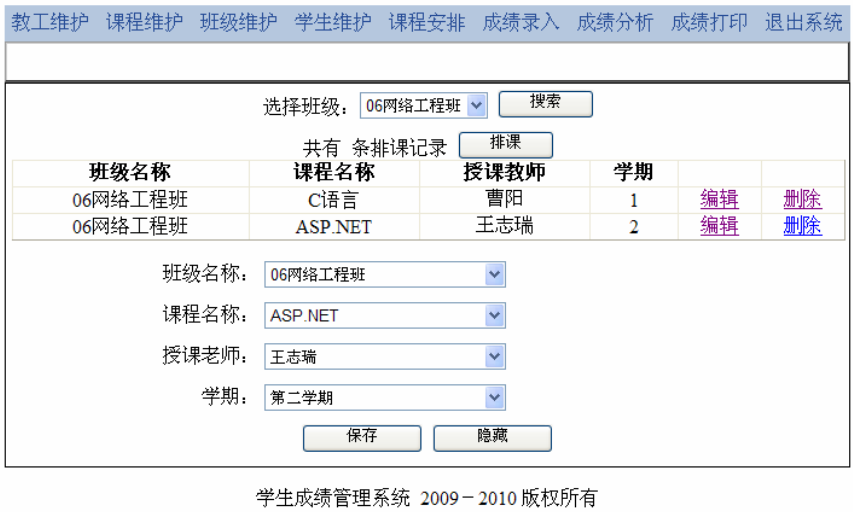


图 1.9　排课信息维护页面

其中，页面中间以表格形式显示已经排课的信息，并可以进行修改和删除。排课信息在表格下面输入，班级已经排课的情况可以在页面上部进行查询。

6．学生成绩录入

学生成绩录入页面如图 1.10 所示。

图 1.10　学生成绩录入页面

其中，页面上部显示班级和课程，成绩在页面的下部录入。

7．学生成绩打印

学生成绩打印页面如图 1.11 所示。页面上部可以选择班级和课程，显示的成绩可以通过工具栏分页显示。选择打印图标实现打印。

图 1.11　学生成绩打印页面

8．学生课程成绩信息查询

学生课程成绩信息查询页面如图 1.12 所示。

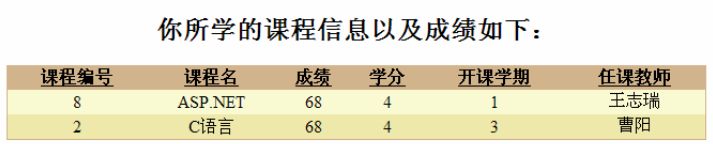


图 1.12　学生课程成绩信息查询页面

1.5　系统结构流程图

前面介绍了系统实现的主要功能，这些功能的关系如图 1.13 所示，主要包含 3 方面。所有人员进入系统都需要登录，学生从登录页面可以进行注册。登录成功进入相应的页面，进行相应的操作。管理员和教师操作结束可以退出系统。

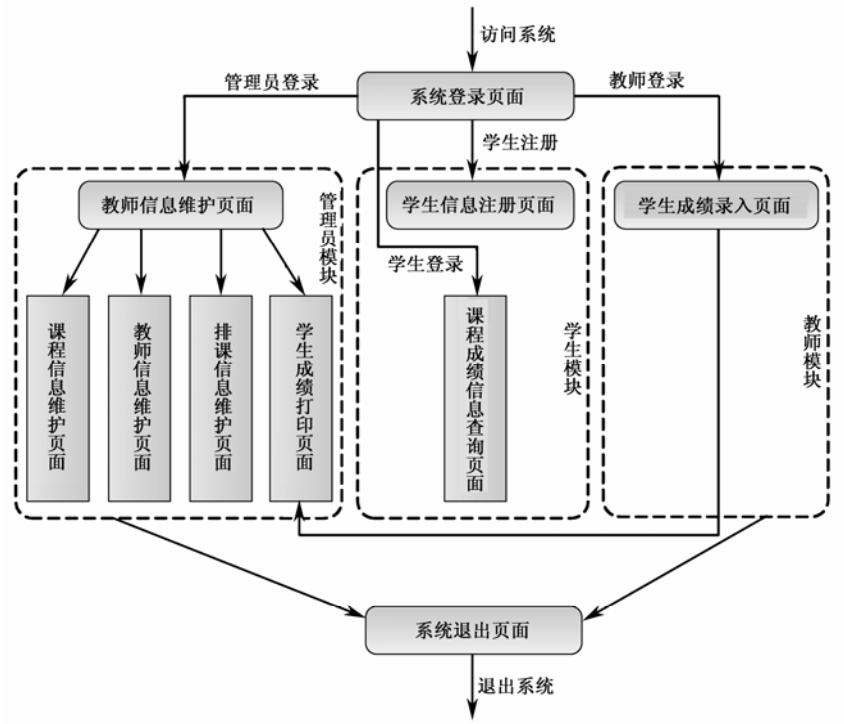


图 1.13　实现的系统结构流程图

第 2 章　创建系统网站项目

◇ 任务目标

根据本项目的功能，需要创建系统所需的网站和系统所需的相关目录结构，为后期的开发做好准备。创建完成后的站点目录结构如图 2.1 所示。

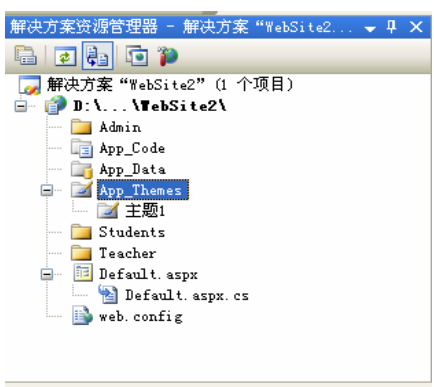


图 2.1　网站目录结构

第一部分　应 用 实 践

2.1　创建系统网站项目

创建系统网站项目步骤如下：

（1）运行 Visual Studio 2008，默认显示出“起始页”，在“起始页”内可以查看微软官方 MSDN 最新的技术文档和最近开发的项目列表。

（2）选择“文件”→“新建”→“网站”菜单项，打开“新建网站”窗口，如图 2.2 所示。

（3）在“新建网站”窗口已安装的模板中选择“**ASP.NET 网站**”模板，在窗口的顶部下拉框内，选择“**.NET Framework 3.5**”框架，在“语言”下拉框中选择“Visual C#”。

（4）单击“浏览”按钮，出现如图 2.3 所示的“选择位置”对话框，有 4 个站点位置选项，默认选项是“文件系统”，“文件系统”方式允许把所要创建的站点文件保存到电脑磁盘的任何位置。选中 D 盘，单击按钮，在 D 盘下创建一个“学生成绩管理系统”文件夹。选中此文件夹，单击“打开”按钮，返回到“新建网站”窗口，此时“新建网站”窗口的

“位置”行中的下拉列表框会选中“文件系统”，对应的路径为“D:\学生成绩管理系统”，单击“确定”按钮，完成站点的创建工作。

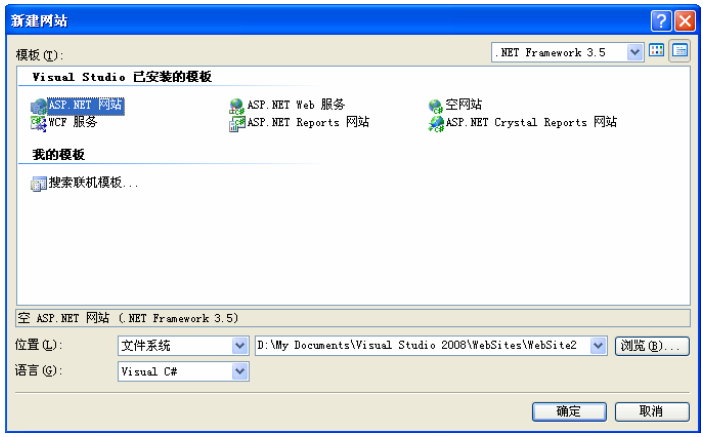


图 2.2　新建网站对话框

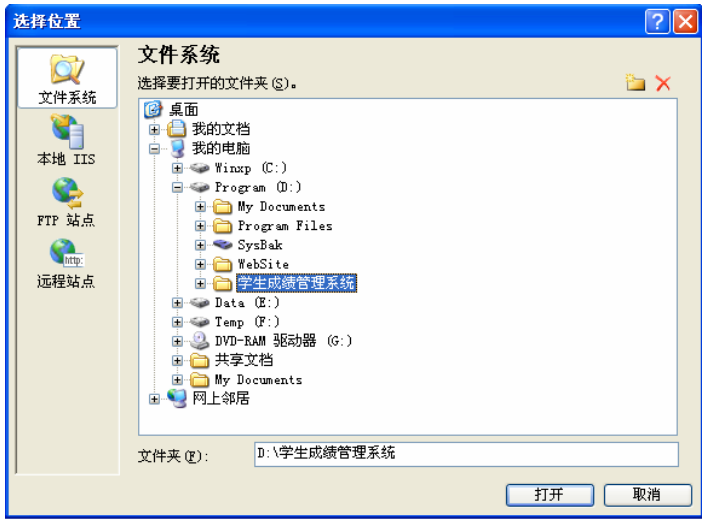


图 2.3　选择位置对话框

（5）创建成功后，默认打开如图 2.4 所示的 Visual Studio 2008 开发环境。

Visual Studio 2008 开发环境主要包括如下几个部分：

①“解决方案资源管理器”窗口：此窗口用来列出本网站内所有的文件资源，方便快速地定位站点内的所有内容。

网站建成后，Visual Studio 2008 会为网站创建一些默认的文件结构，在“**解决方案资源管理器**”窗口内可以看到如下内容：一个“**web.config**”文件，一个“App_Data”文件夹，一个“Default.aspx”文件和此文件的**代码分析文件**“Default.aspx.cs”。

②**“属性”窗口**：此窗口用来对 Web 窗体内选中的控件或标签的属性进行快速地可视化设置。

③**“工具箱”窗口**：此窗口用来列出在 Visual Studio 2008 内可被 Web 窗体使用的所有控件，方便开发者查找使用。

④**“Web 窗体”设计窗口**：此窗口提供三种视图来对 Web 窗体进行设计。

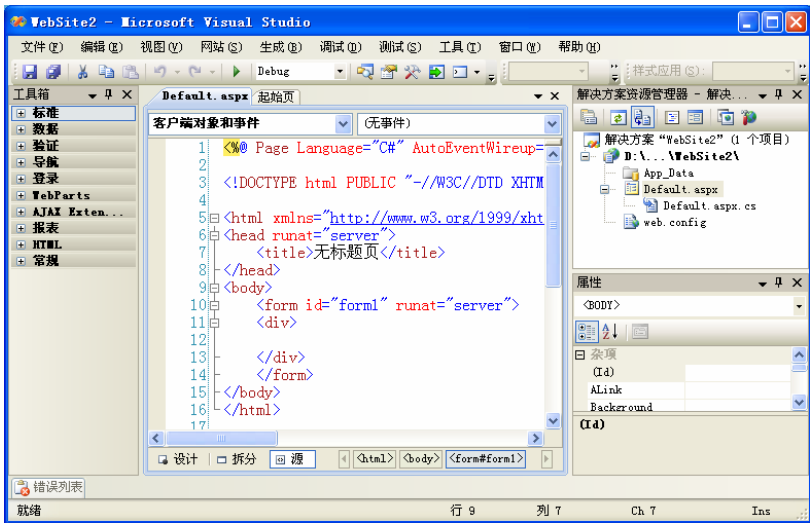


图 2.4　Visual Studio 2008 开发界面

2.2　创建系统所需目录

学生管理系统从参与者的角度把整个系统分为 3 个主要模块，为了便于不同模块之间的文件管理，需要在网站内为不同的模块创建不同的文件夹。所创建的文件夹名称如下：

管理员：Admin

教师：Teacher

学生：Students

ASP.NET 提供了一些**特殊的文件夹**，用来存放特定的内容，在项目中主要用到如下几个特殊文件夹：

App_Data

App_Code

App_Themes

（1）在“解决方案资源管理器”窗口中，右击“网站根目录”，选择“新建文件夹”，按照相同的方法分别创建“Admin”、“Teacher”、“Students”文件夹。

（2）在“解决方案资源管理器”窗口中，右击“网站根目录”，选择“添加 ASP.NET 文件夹”，如图 2.5 所示。分别选中其中的“App_code”和“主题”，对应的文件夹就会自动添加到当前项目中。创建结束后，“解决方案资源管理器”窗口内的目录结构如图 2.1 所示。

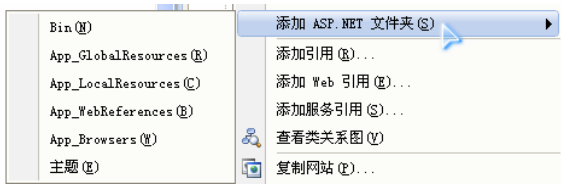


图 2.5　ASP.NET 特殊文件夹

第二部分　知识点链接

L2.1　创建系统网站项目

L1.　ASP.NET网站

Visual Studio 2008 是.NET 环境下功能比较完备的开发平台，可以用来开发很多种项目，为不同的项目提供了不同的项目模板。需要建立网站项目，可以通过“文件”→“新建”→“项目”菜单项，打开“新建项目”窗口，如图 2.6 所示，在此窗口左侧选择“Web”，右侧会出现 ASP.NET 支持的项目类型，选择其中的“ASP.NET Web 应用程序”。同样也可以通过“文件”→“新建”→“网站”菜单项，快速地创建 ASP.NET 网站项目。

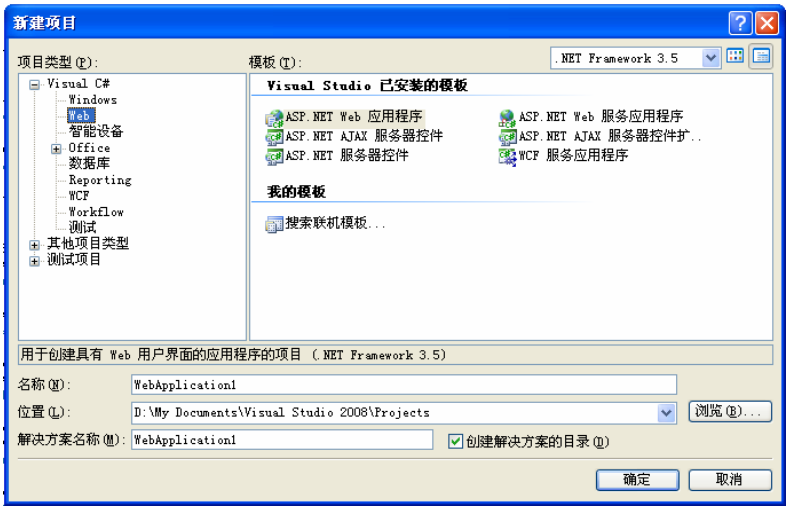


图 2.6　Visual Studio 2008 Web 项目模板

L2. .NET Framework 3.5

.NET Framework 是 Microsoft 为开发应用程序创建的一个富有革命性的新平台，是微软下一代平台（Dot Net）的支持库。使用.NET 开发的程序需要在 .NET Framework 下才能运行。

.NET FrameWork 的体系结构包括 5 大部分，它们是：

① 程序设计语言及公共语言规范（CLS）。

② 应用程序平台（ASP.NET 及 Windows 应用程序等）。

③ ADO.NET 及类库。

④ 公共语言运行时（CLR）。

⑤ 程序开发环境（Visual Studio）。

其结构如图 2.7 所示。

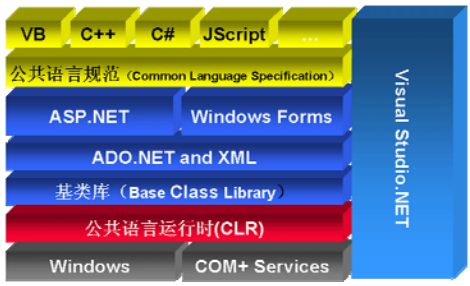


图 2.7　.NET 框架结构

构建在 Windows 操作系统之上的是公共语言运行时（CLR），其作用是负责执行程序，提供内存管理、线程管理、安全管理、异常处理、通用类型系统与生命周期监控等核心服务。在 CLR 之上的是.NET Framework 类库，提供许多类与接口，包括 ADO.NET、XML、IO、网络、调试、安全和多线程等。

.NET Framework 类库以命名空间（Namespace）方式组织类库，命名空间与类库的关系就像文件系统中的文件夹与文件的关系一样，例如，用于处理文件的类属于 System.IO 命名空间。

在.NET 框架基础上的应用程序主要包括 ASP.NET 应用程序和 Windows Forms 应用程序，其中 ASP.NET 应用程序又包含了 Web Forms 和 Web Service，它们组成了全新的因特网应用程序；而 Windows Forms 是全新的窗口应用程序。

在.NET Framework 之上，无论采用哪种编程语言编写的程序，都被编译成中间语言 IL，IL 经过再次编译形成机器码，完成 IL 到机器码编译任务的是 JIT（Just In Time）编译器。上述处理过程如图 2.8 所示。

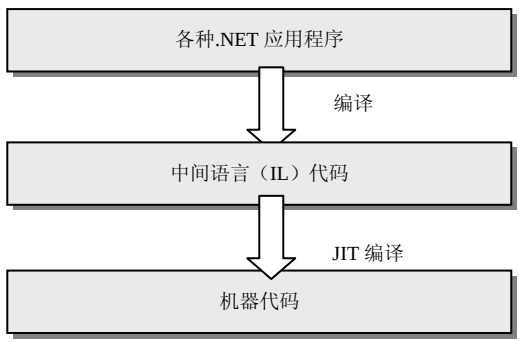


图 2.8　.NET 应用程序的编译过程

对于 ASP.NET 应用程序，使用 IL 和 JIT 技术还能够提高执行效率。当第一次执行 ASP.NET 程序时，它先被编译为中间语言代码，再由 JIT 编译器将中间语言代码编译为机器码，并将机器码存放在缓存中。以后再执行该程序时，只要程序没有变化，系统将直接从缓存中读取机器码，从而大大提升了执行效率。

随着.NET 技术的不断发展，.NET Framework 的发展也经历了几个阶段，从早期的.NET Framework 1.0、1.1，发展到 .NET Framework 2.0，标示着.NET 技术走向成熟，功能更加强大。2008 年，随着 Microsoft 推出 Visual Studio 2008 开发平台，.NET Framework 又由 2.0 更

新为 3.0、3.5。.NET Framework 3.0、3.5 在.NET Framework 2.0 的基础上进行扩展，增加了很多新特性，如 WCF、WPF、WF、LINQ 和 AJAX 等，使.NET 技术更加强大、成熟。在利用 Visual Studio 2008 进行项目开发的时候，可以根据所要使用的特性选择所采用的.NET Framework 版本，实现与早期版本的兼容。

L3. 选择位置对话框

网站是管理应用程序并向外发布信息的基本单位，也是网站迁移的基本单位。在 ASP.NET 中，一个网站就是一个应用程序。由于应用目的的不同，Visual Studio 2008 提供了 4 种存放文件的选择方式：

① 文件系统；

② 本地 IIS；

③ FTP 站点；

④ 远程站点。

选择“文件”→“新建网站”菜单项，单击“浏览”按钮，会打开“选择位置”窗口，如图 2.9 所示。

下面就每种的特点进行简要的介绍说明。

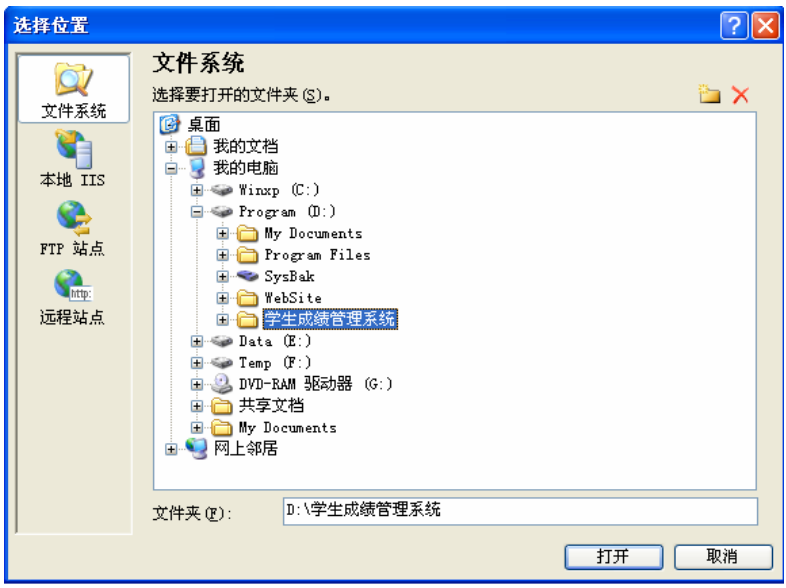


图 2.9　选择位置对话框

1．文件系统网站

文件系统网站是一种用于检验和调试的网站，只能用来检验和调试应用程序，而不能向外发布信息。文件系统网站的文件夹可以放置在本机任意物理目录下，因此非常适合调试或学习时使用。

使用文件系统网站时，并不需要在计算机上安装 IIS 服务器。此时系统将自动为该网站配置一个开发服务器（ASP.NET Development Server），用来模拟 IIS 服务器对网站运行时的支持。开发服务器是一种轻量级服务器，它并不具备 IIS 的全部服务功能，但在通常情况

下，利用它进行调试已经够用。当使用文件系统网站时，系统会自动调用开发服务器来调试运行的网页，同时给网站随机地分配一个端口。例如，调试的网页名是 MyPage.aspx，当运行开发服务器时，该网页的 URL 是“http://localhost:12345/[网站名]/MyPage.aspx”。其中网站名就是应用程序的根目录名。12345 在这里只是一个示例，它是开发服务器给应用程序随机生成的一个端口。

2．本地IIS网站

如果机器上安装有 IIS 服务器，就可以创建本地 IIS 网站。此时的网站文件夹必须直接或间接地放在虚拟目录下面。

创建本地 IIS 网站的步骤是：

（1）在打开的“新建网站”对话框的“位置”下拉列表框中选择 HTTP。

（2）单击“浏览”按钮，打开“选择位置”对话框。

（3）在“选择位置”对话框的左侧选择“本地 IIS”，再选中右侧的“默认网站”，最后在右边选择两个图标之一：其中靠左的是“创建新 Web 应用程序”图标；靠右的是“创建新虚拟目录”图标。前者用于直接在虚拟目录下创建网站，后者用于创建一个指向另一物理目录的虚拟目录。对话框如图 2.10 所示。

（4）如果选择“创建新虚拟目录”图标，还需要在打开的对话框中设置虚拟目录名（即别名），以及对应的物理目录名，如图 2.11 所示。

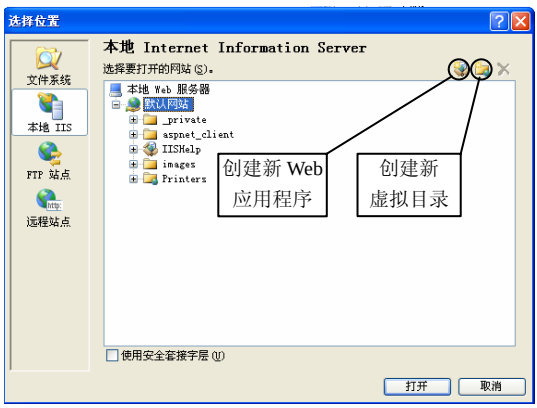


图 2.10　选择建立虚拟目录的方式

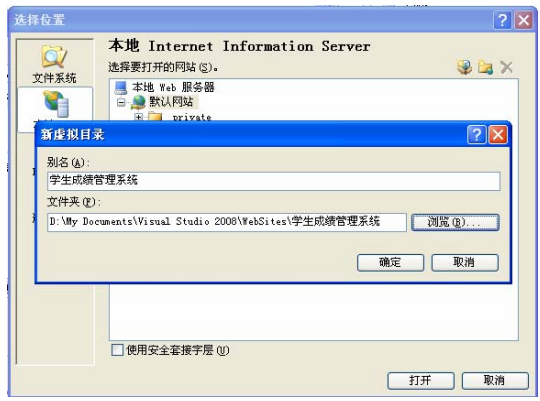


图 2.11　新建虚拟目录的方式

3．FTP站点

FTP 站点方便在创建网站的时候把项目文件保存在远程 FTP 站点之上，可以同时被多人访问使用，方便大企业统一管理的目的。但是 Visual Studio 中的 FTP 站点并没有提供代码的版本管理功能，因此需要进行团队开发，需要采用 VSTS 或者专业的版本控制系统进行项目的开发配置管理。

4．远程站点

远程站点是可以向外发布信息的网站，一个远程网站必须获得唯一的 URL 地址（并且安装有扩展的 FrontPage）。为了将调试好的网站传送到远程网站，可以利用 FTP 文件服务器，将调试好的网站用字符流的方式传送到远程网站的指定目录中。为此，必须获得远程网

站的允许并取得相应的协议，才可以进行此项传输工作。

L4. web.config

web.config 是 ASP.NET 网站项目中很重要的一个配置文件。web.config 文件是一个 XML 文本文件，用来储存 ASP.NET Web 应用程序的配置信息（如规定客户的认证方法，基于角色的安全技术的策略，数据绑定的方法，错误显示方式，数据库连接串等），它可以出现在应用程序的每一个文件夹中。当通过 Visual Studio 2008 新建 Web 应用程序后，默认情况下会在根目录自动创建一个默认的 web.config 文件，包括默认的配置设置，所有的子目录都继承它的配置设置。如果想修改子目录的配置设置，可以在该子目录下新建一个 web.config 文件。它可以提供除从父目录继承的配置信息以外的配置信息，也可以重写或修改父目录中定义的设置。

在运行时对 web.config 文件的修改不需要重启服务器就可以生效。当然 web.config 文件是可以扩展的，可以自定义新配置参数并编写配置节处理程序，以对它们进行处理。

web.config 配置文件（默认的配置设置）所有的代码都应该位于

```
<configuration>
<system.web>
```

和

```
</system.web>
</configuration>
```

之间，其中主要包括如下几个配置节。

1．<authentication>

作用：配置 ASP.NET 身份验证支持（Windows、Forms、PassPort、None）。该元素只能在计算机、站点或应用程序级别声明。<authentication>元素必须与<authorization> 节配合使用。

示例：以下示例为基于窗体（Forms）的身份验证配置站点，当没有登录的用户访问需要身份验证的网页时，网页自动跳转到登录网页。

```
<authentication mode="Forms" >
    <forms loginUrl="logon.aspx" name=".FormsAuthCookie"/>
</authentication>
```

其中元素 loginUrl 表示登录网页的名称，name 表示 Cookie 名称。

2．<authorization>

作用：控制对 URL 资源的客户端访问（如允许匿名用户访问）。此元素可以在任何级别（计算机、站点、应用程序、子目录或页）上声明。

示例：以下示例禁止匿名用户的访问。

```
<authorization>
    <deny users="?"/>
</authorization>
```

3．<customErrors>

作用：为 ASP.NET 应用程序提供有关自定义错误信息的信息。它不适用于 XML Web Services 中发生的错误。

示例：当发生错误时，将网页跳转到自定义的错误页面。

```
<customErrors defaultRedirect="ErrorPage.aspx" mode="RemoteOnly">
</customErrors>
```

其中元素 defaultRedirect 表示自定义的错误网页的名称。mode 元素表示对不在本地 Web 服务器上运行的用户显示自定义（友好的）信息。

4．<pages>

作用：标识特定于页的配置设置（如是否启用会话状态、视图状态，是否检测用户的输入等）。可以在计算机、站点、应用程序和子目录级别声明。

示例：不检测用户在浏览器输入的内容中是否存在潜在的危险数据（注：该项默认是检测，如果使用了不检测，一定要对用户的输入进行编码或验证），在从客户端回发页时将检查加密的视图状态，以验证视图状态是否已在客户端被篡改。

```
<pages buffer="true" enableViewStateMac="true" validateRequest="false"/>
```

5．<sessionState>

作用：为当前应用程序配置会话状态设置（如设置是否启用会话状态，会话状态保存位置）。

示例：

```
<sessionState mode="InProc" cookieless="true" timeout="20"/></sessionState>
```

mode="InProc"：表示在本地储存会话状态（也可以选择储存在远程服务器或 SAL 服务器中，或不启用会话状态）。

cookieless="true"：表示如果用户浏览器不支持 Cookie 时启用会话状态（默认为 false）。

timeout="20"：表示会话可以处于空闲状态的分钟数。

6．<trace>

作用：配置 ASP.NET 跟踪服务，主要用来程序测试判断哪里出错。

示例：以下为 Web.config 中的默认配置：

```
<trace enabled="false" requestLimit="10" pageOutput="false" traceMode="SortByTime" localOnly="true" />
```

enabled="false"表示不启用跟踪。requestLimit="10"表示指定在服务器上存储的跟踪请求的数目。pageOutput="false"表示只能通过跟踪实用工具访问跟踪输出。traceMode="SortByTime"表示以处理跟踪的顺序来显示跟踪信息。localOnly="true" 表示跟踪查看器（trace.axd）只用于宿主 Web 服务器。

L5. 代码分离文件

每个 ASPX 网页中包含两方面的代码：用于定义页面外观的代码和用于处理页面后台逻辑的代码。用于定义页面外观的代码主要包括 HTML 标记、CSS 代码，以及对 Web 服务

器控件的定义等；用于处理页面后台逻辑的代码主要是用 C# 或其他语言编写的事件处理程序。

在 ASP.NET 中，这些代码可以用两种模式存储：一种是代码分离模式，另一种是单一模式。在代码分离模式中，定义页面外观的代码与处理页面后台逻辑的代码分别放在不同的文件中（.aspx 和.aspx.cs）；在单一模式中，将两种代码放置在同一个文件中（.aspx）。

新建 ASPX 网页时可以选择代码存储模式，设置的方法如图 2.12 所示。

图 2.12　选择代码存储模式

对话框右下角的第一个复选框用来确定存储模式。如果被选中，将使用分离模式；否则使用单一模式。推荐使用代码分离模式，这种模式代码结构清晰且便于调试和维护，适合大型项目的代码规范要求，本书后续章节的综合示例程序将采用这种模式。

L6. 解决方案资源管理器

解决方案资源管理器提供项目及其文件的有组织的视图，并且提供对项目和文件相关命令的便捷访问。与此窗口关联的工具栏提供适用于列表中突出显示的项的常用命令。若要访问解决方案资源管理器，可在“视图”菜单上选择“解决方案资源管理器”。

L7. 工具箱窗口

ASP.NET 提出了一种全新的 B/S 系统开发模式，是一种类似 GUI 的基于事件触发的开发模式，因此在 ASP.NET 中根据功能的不同提供了很多不同种类的控件。ASP.NET 所提供的内置控件可以通过工具箱窗口方便地查看与使用。

根据控件类别和功能的不同，分为如下几组：

HTML：此组内的控件代表的是浏览器端 HTML 的标记（除此之外的分组都是服务器控件），方便网页开发过程中快速地创建 HTML 标记，但是并不是所有的 HTML 标记都在此组内，其中只包含了部分常用的 HTML 标记。

标准：此组内的控件是 ASP.NET 提供的基本服务器控件，内部包含了很多基本服务器控件，在网页开发过程中是必不可少的控件，因此是一组非常重要的控件。

数据：此组内的控件是 ASP.NET 提供的便利的数据源控件和各种用途的数据绑定控

件，是在系统开发过程中经常需要被使用到的控件，是非常重要的一些控件。

验证：此组内的控件主要用来对基本控件提供数据验证功能，方便快速地实现表单数据的验证，是一组很重要的控件。

导航：此组内的控件主要用来实现页面导航功能，在系统开发过程中也必不可少，是一种很重要的控件。

WebParts：此组内的控件主要用来为创建动态的网页提供支持，利用它们可以快速地开发出允许用户进行配置和个性化的页面（用户可以方便地显示、隐藏、移动相应的 WebParts 组件）。

AJAX Extensions：此组内的控件主要用来开发 AJAX 应用，通过这些控件，可以很方便地开发出支持局部更新的页面，提高了 AJAX 应用程序的开发效率。

报表：此组内的控件是对水晶报表工具的封装，利用它们可以快速地把水晶报表应用于 Web 开发中。

L8. Web窗体设计窗口

在 Visual Studio 2008 中，Web 窗体设计窗口比 Visual Studio 2005 有了改进。在 Visual Studio 2008 中，针对.aspx 文件提供了“设计”窗口、“源”窗口和“拆分”窗口。“设计”窗口方便初学者可视化地对页面的外观进行设计；“源”视图方便对 XHTML 代码比较熟悉的开发人员进行细微的页面调整；“拆分”窗口能够把“设计”窗口与“源”窗口同时显示出来，方便开发人员同时使用“设计”视图和“源”视图来对页面的外观进行设计。

L2.2 创建系统所需目录

L1. 特殊的文件夹

程序员在设计程序时应该将特定类型的文件存放在某些文件夹内，以方便管理和操作。ASP.NET 保留了一些文件和文件夹名称，程序员可以直接使用。不同的文件夹存放不同类型的文件，下面分别对这些文件夹进行介绍。

1．App_Data 文件夹

该文件夹包含应用程序数据文件，包括 MDF 文件、XML 文件和其他数据存储文件。ASP.NET 使用 App_Date 文件夹存储应用程序的本地数据库，该数据库可用于维护成员资格和角色信息，其他数据库也可以放在该文件下。

2．Bin文件夹

该文件夹包含要在应用程序中引用的控件、组件或其他代码的已编译程序集（.dll 文件）。通常，该文件夹不需要手动添加，在运行创建的 Web 应用程序时会自动生成。

3．App_Themes文件夹

该文件夹包含用于定义 ASP.NET 页面和控件外观的文件集合（.skin 和.css 文件，以及图片文件和一些资源文件）。

4．App_Browsers文件夹

不同的浏览器或相同浏览器的不同版本支持不同的功能。在应用程序中，可能需要确

定用户正在使用哪种类型的浏览器查看本页，并可能需要确定浏览器是否支持某些特定功能。ASP.NET 需要用扩展名为.browser 的文件保存这些特殊浏览器的定义信息。.browser 文件就存放在该文件夹中。

5．App_GlobalResources文件夹

该文件夹包含编译到具有全局范围的程序集的资源（.resx 和.resources 文件）。此文件夹中的资源是强类型的，可以通过编程方式进行访问。

6．App_LocalResources文件夹

该文件夹包含与应用程序中的特定页、用户控件或母版页关联的资源（.resx 和.resources 文件）。

7．App_WebReferences文件夹

该文件夹包含用于定义在应用程序中和 Web 引用相关的文件。

第3章 设计数据库结构与构建访问层

◇ 任务目标

本章主要目标是完成学生成绩管理系统所需数据库的设计，以及构建系统所需的数据访问层和系统常用的功能模块，把常用的数据库操作接口和相关的功能封装成可以直接调用的类和方法，在后期项目中只需通过简单的代码调用就可以完成复杂的功能，提高系统后期开发的效率，并减轻程序员开发的复杂度。

第一部分 应 用 实 践

3.1 设计系统数据库结构

成绩管理系统的数据库采用弱冗余的原则，每张表和每个字段都强调规范性。本书主要通过 SQL Server 实现数据库的总体设计和具体的表设计。

根据成绩管理系统的功能模块，以及数据库之间的约束关系，成绩管理系统的数据库主要设计了 6 张表。数据库名称为 SMS，内部的表名和字段名分别如下：

（1）班级表（Class）：班级编号、班级名称。

（2）学生表（Student）：学号、姓名、性别、年龄、所属班级编号、手机、邮箱、入学年份、学制、家庭地址、登录密码、个人爱好、个人图片。

（3）教工表（Teacher）：教工编号、姓名、性别、手机、座机、邮箱、登录密码。

（4）课程表（Course）：课程编号、课程名、课程学分。

（5）课程安排表（Arrange）：班级编号、课程编号、成绩是否录入、开课学期、授课老师编号。

（6）学生成绩表（Score）：学号、课程编号、课程成绩。

表作用说明、详细表结构、表内字段说明，详见表 3.1～表 3.6。

教工表用来保存所有教工的信息。teaID 字段的自动生成的设置方法是在 teaID 字段属性窗口中“(是标识)”设置为“是”，“标识增量”设置为“1”，“标识种子”设置为“1”。

班级表用来保存所有班级的信息，其中 classNums 字段的默认值的设置方法是在 classNums 字段属性窗口中“默认值或绑定”设置为“0”。

班级人数通过数据库触发器自动维护。

表 3.1　教工表结构（Teacher）

字 段 名 称	数 据 类 型	功 能 说 明
teaID（主键）	int	教工编号（自动生成）
teaName	nchar(10)	教工姓名
teaSex	nchar(1)	教工性别，取值：F(0) M(1)
teaTelephone	nchar(11)	教工手机
teaPhone	nvarchar(20)	教工座机
teaType	nchar(1)	教工类别：系统管理员(0)或教师(1)
teaEmail	nvarchar(50)	教工邮箱
teaPassword	nchar(10)	教工登录系统所用密码

表 3.2　班级表结构（Class）

字 段 名 称	数 据 类 型	功 能 说 明
classID（主键）	nchar(6)	班级编号
className	nchar(50)	班级的名称
classNums	int	班级人数（通过触发器维护，默认值为 0）
enrollTime	datetime	入学时间
gradTime	datetime	毕业时间
classCounselor	int	班级辅导员的编号

表 3.3　课程表结构（Course）

字 段 名 称	数 据 类 型	功 能 说 明
courseID（主键）	int	自动编号，自动生成
courseName	nchar(10)	课程名称
courseScore	int	课程学分

课程表用来保存所有课程的信息。courseID 列的自动编号的设置方法是在 courseID 列属性中“(是标识)”设置为“是”，“标识增量”设置为“1”，“标识种子”设置为“1”。

表 3.4　学生表结构（Student）

字 段 名 称	数 据 类 型	功 能 说 明
stuID （主键）	nchar(8)	学生学号
stuName	nchar(10)	学生姓名
stuSex	nchar(1)	学生性别，取值：F(0) M(1)
stuAge	int	年龄
stuClassID	nchar(6)	学生所属班级编号
stuTelphone	nchar(11)	学生手机号码
stuEmail	nvarchar(100)	学生电子邮箱
stuEnrollmentTime	nchar(4)	学生入学年份
stuHomeAddress	nvarchar(100)	学生家庭住址
stuPassword	nvarchar(60)	学生登录系统所用密码
stuPic	nvarchar(200)	学生个人一寸照片（存放图片的相对路径）
stuLove	nvarchar(100)	学生个人爱好

学生表用来存储所有的学生信息。

表 3.5　课程安排表结构（Arrange）

字 段 名 称	数 据 类 型	功 能 说 明
classID	nchar(6)	班级编号
courseID	int	课程编号
isRecord	nchar(1)	成绩是否录入（未录入为 0，录入为 1）
team	nchar(1)	上课学期（1,2,3,4,5,6）
teaID	int	教师编号

课程安排表用来存储所有的课程安排信息。

表 3.6　学生成绩表结构（Score）

字 段 名 称	数 据 类 型	功 能 说 明
stuID	nchar(8)	学生编号
courseID	int	课程编号
score	int	所获得的成绩

学生成绩表用来存储学生的所有课程的成绩信息。

不同表之间还存在着内部的关联关系，对表内数据的修改需要满足它们之间的这些关联约束，具体的数据库表之间的关联关系如图 3.1 所示。

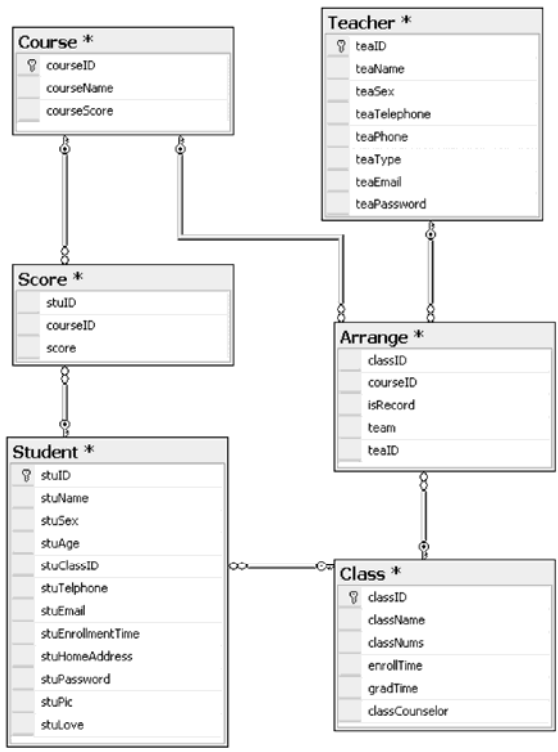


图 3.1　数据表之间关系

3.2　系统数据库实现

本实例将使用微软的 SQL Server 2005 作为数据库服务器。本节主要介绍如何利用 SQL Server 2005 建立如上的数据库及相关的数据表。

（1）选择“开始”→“所有程序”→“Microsoft SQL Server 2005”→“SQL Server Management Studio”，出现如图 3.2 所示的界面，从“服务器名称”下拉列表中选择或输入所要连接的 SQL Server 2005 服务器名称，“身份验证”下拉框中选择“SQL Server 身份验证”，并在“登录名”和“密码”框处输入对应的登录名和密码，单击“连接”按钮，出现如图 3.3 所示的界面。

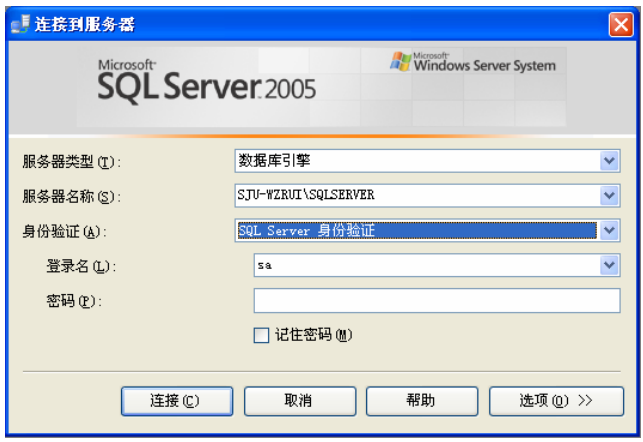


图 3.2　SQL Server 2005 登录界面

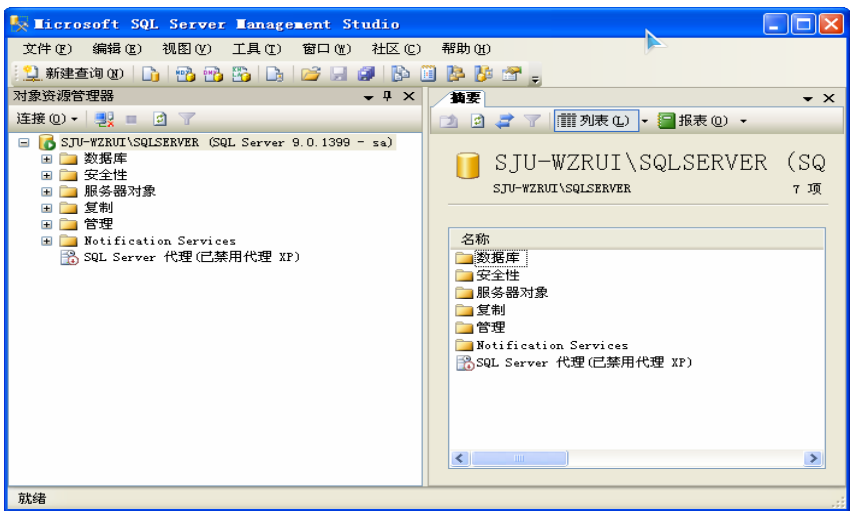


图 3.3　SQL Server 2005 服务器管理界面

（2）右击分支中的“数据库”，选择“新建数据库”，出现如图 3.4 所示的界面。在“数据库名称”处输入“SMS”，单击“确定”按钮，即可建立好一个空的 SMS 数据库，展开“数据库”目录，就会看到“SMS”数据库，如图 3.5 所示。

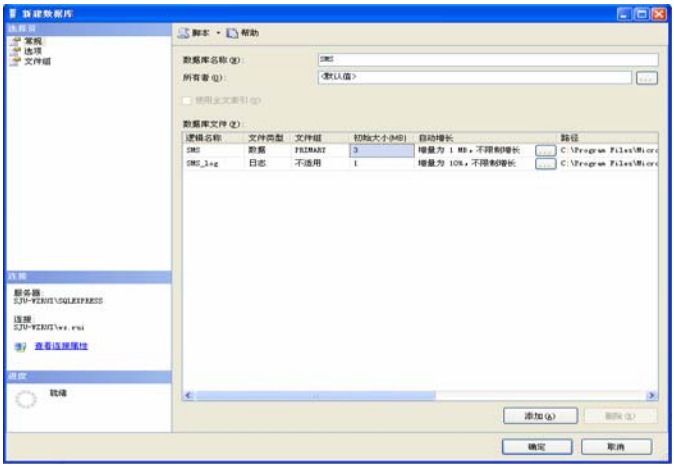


图 3.4　界面方式新建数据库

图 3.5　建成后的 SMS 数据库

（3）展开“SMS”目录，右击“表”节点，选择“新建表”，出现表设计界面。在“列名”处输入对应的字段，在“数据类型”处选择对应的数据类型，并对有关的数据类型进行相应的设置。所有的输入结束后，右击要设为主键的列，选择“设置主键”，然后单击“保存”按钮，在弹出的窗口中输入“Student”即可设计 Student 表，设计如图 3.6 所示。其他表的建立过程类似，请自行建立。

注：这里以建立 Student 表为例演示，其他表的建立过程类似。

列名	数据类型	允许空
stuID	nchar(8)	☐
stuName	nchar(10)	☑
stuSex	nchar(1)	☑
stuAge	int	☑
stuClassID	nchar(6)	☑
stuTelphone	nchar(11)	☑
stuEmail	nvarchar(100)	☑
stuEnrollmentTime	nchar(4)	☑
stuHomeAddress	nvarchar(100)	☑
stuPassword	nvarchar(60)	☑
stuPic	nvarchar(200)	☑
stuLove	nvarchar(100)	☑
		☐

图 3.6　表设计界面

（4）右击“SMS”，选择“新建查询”，通过 **SQL 语句**的方式为所创建的表插入一些初始数据，代码如下：

```
INSERT INTO Class(classID,className,enrollTime,GradTime)
VALUES('B06051','06 网络工程班','2006-09-01','2010-07-01')
INSERT INTO Class(classID,className,enrollTime,GradTime)
VALUES('B06053','06 软件工程班','2006-09-01','2010-07-01')
……
```

在开发过程中所需要的测试数据，可以通过这种方式自行添加。

（5）添加表的外键约束。在数据库的“对象资源管理器”中右击 SMS 数据库中的“数据库关系图”，选择“新建数据库关系图”选项，在弹出的“添加表”对话框中添加所有的表，右击“Score”表，选择“关系”选项，在弹出的“外键关系”对话框中单击“表和列规范”后面出现... 图标，弹出“表和列”对话框。在此对话框中主键表选择“Course”，选择主键“CourseID”；外键表选择“Score”，选择“CourseID”。这样“Course”和“Score”的外键关系设置成功。其他的表关系设置步骤与其类似，设置好后如图 3.1 所示。

（6）班级表中的班级人数字段，此字段的值应根据学生表内学生的添加或删除操作自动进行增加或减少，因此需要为其创建一个触发器，让其根据学生表内学生记录的变化自动维护。创建过程如下：展开“SMS”数据库文件夹，展开“表”文件夹，展开“Student”，右击“触发器”，选择“新建触发器”，打开“新建触发器”窗口，本窗口已生成了新建触发器的模板代码，可以在此基础上进行修改，修改完成后的触发器代码如下：

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
//为“添加”创建触发器
CREATE TRIGGER student_insert ON Student FOR INSERT
AS
BEGIN
    DECLARE @num INT
    SELECT @num = c.classNums FROM Class c INNER JOIN INSERTED i ON c.classID = i.stuClassID
    UPDATE Class SET classNums = @num+1    FROM Class c
    INNER JOIN INSERTED    i ON    c.classID = i.stuClassID
END
GO
//为“修改”创建触发器
CREATE TRIGGER student_delete ON Student FOR DELETE
AS
BEGIN
    DECLARE @num INT
    SELECT @num = c.classNums FROM Class c INNER JOIN DELETED d
            ON c.classID = d.stuClassID
    UPDATE Class SET classNums = @num-1    FROM Class c
INNER JOIN DELETED    d ON    c.classID = d.stuClassID
END
GO
```

（7）由于系统中经常进行学生插入操作，为了提高执行效率，可以为此操作创建一个存储过程。创建过程如下：展开“SMS”数据库文件夹，展开“可编程性”文件夹，右击“存储过程”，选择“新建存储过程”，打开“创建存储过程”窗口，本窗口已生成了新建存

储过程的模板代码，可以在此基础上进行修改，修改完成后的存储过程的代码如下：

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE insertStudent
@stuID NCHAR(8),@stuName NCHAR(20),@stuSex NCHAR(1),@stuAge int,
@stuClassID NCHAR(8),@stuEmail NVARCHAR(100),
@stuTelphone NvarCHAR(11),@stuEnrollmentTime NCHAR(4),
@stuHomeAddress NVARCHAR(100),@stuPassword NCHAR(32),
@stuLove NVARCHAR(100),@stuPic NVARCHAR(200)
AS
    INSERT INTO Student
    (stuID,stuName,stuSex,stuAge,stuClassID,stuEmail,stuTelphone,stuEnrollmentTime,stuHomeAddress,
      stuPassword,stuLove,stuPic)VALUES(@stuID,@stuName,@stuSex,@stuAge,@stuClassID,@stuEmail,
      @stuTelphone,@stuEnrollmentTime,@stuHomeAddress,@stuPassword,@stuLove,@stuPic)
```

3.3 构建数据访问层

数据访问层主要任务是对下层实现对数据源的检索和编辑功能，对上层提供一个方便调用的数据访问接口，调用者只需传递必须的参数即可实现数据库的检索和操作功能。

在数据访问层的内部把常用的数据库访问方法封装，把重复的内容封装在类和方法的内部，把需要用户提供的内容通过参数的方式传入，之后，用户需要对数据源进行检索或编辑的时候，只需调用封装好的数据访问层即可。这样既方便、简化了开发人员的开发和学习的复杂度，又提高了后期软件维护的效率。

下面通过边编写边解释的方式完成数据库访问层。

（1）运行 Visual Studio 2008，进入 Visual Studio 2008 起始页，选择“文件”→“打开”→“网站”菜单项，打开前面创建的“学生成绩管理系统”网站，展开“解决方案资源管理器”窗口，右击空白处，选择“添加”→“新建项”菜单项，打开“添加新项窗口”对话框，选择“类”模板，并命名类名为“commDBHelper.cs”。

（2）单击“添加”按钮，提示“是否把文件放入到 APP_Code 目录中”，单击“是”按钮，进入到此类的代码窗口，利用此类实现数据访问层。

（3）把数据库连接串配置到 web.config 文件中。由于便于系统的移植、安全等原因，通常需要把数据源的连接串保存在 web.config 配置文件的<connectionStrings>配置节中，因此需要首先在 web.config 文件中把<connectionStrings/>改成如下的形式：

```
<connectionStrings>
        <add name="SqlServer2005ConnectionString" connectionString="DataSource=
            SJU-WZRUI\SQLSERVER;Initial Catalog=SMS;User ID=sa;Password=9ol."
            providerName="System.Data.SqlClient"/>
</connectionStrings>
```

其中主要由 3 个属性组成：name 表示连接串名称；connectionString 表示数据源连接串（这里以连接 SQL Server 2005 为例），DataSource 为数据库服务器的名称，Initial Catalog 为要连接的数据库名称，User ID 和 Password 为数据库系统所需要的用户名和密码信息；providerName 表示数据源提供程序。

此数据库连接也可以使用数据源控件生成，操作方法如下：

① 打开 Default.aspx“设计”视图，从工具箱中拖放一个 SqlDataSource 控件到此页面，单击右上角“SqlDataSource 任务”中的“配置数据源”，在弹出的“配置数据源”对话框中单击“新建连接”按钮。

② 在弹出的“添加连接”对话框中，“数据源”选择“Microsoft SQL Server”，“服务器名”选择要连接的服务器。选择“使用 SQL Server 身份认证”登录到服务器，其中“用户名”和“密码”分别输入登录到 SQL Server 服务器的用户名和密码。在“选择和输入一个数据库名”的下拉框中选择本系统所创建的数据库 SMS，单击“测试连接”按钮，判断是否连接成功，若连接成功单击“确定”按钮。

③ 返回到“配置数据源”窗口，单击“下一步”按钮，在弹出的“把连接字符串保存到应用程序中”对话框中选中“把此连接另存为”复选框，并命名为“SqlServer2005 ConnectionString”，将用于数据库连接的串保存到应用程序配置文件（web.config）中，打开配置文件则可以看到系统自动生成的数据库连接字符串。

④ 删除 Default.aspx 页面中的 SqlDataSource 控件。

（4）引用相关命名空间。ADO.NET 内部提供了 4 种数据访问程序，不同的数据访问程序包含在不同的命名空间内，其中 SqlServer 数据提供程序包含在 System.Data.SqlClient 内，因此需要在类的顶部引用此命名空间。

```
using System.Web.Configuration;                    //ASP.NET 中用于读取 web.config 的命名空间
//ADO.NET 中用于访问 SQL Server 2005 数据的数据提供程序的命名空间
using System.Data.SqlClient;
using System.Data;                                 //引入数据操作命名空间，能够使用如 DataSet 等数据集类
```

（5）在类的内部定义一个字符串，用来保存所要连接的数据源连接串。数据库连接串通常保存在 Web.config 中，在 ASP.NET 程序中，通过 WebConfigurationManager 类可以实现对 Web.config 配置文件内数据库连接串的读取操作。

```
//从 Web.config 文件中把 SqlServer2005ConnectionString 串的值读取出来，存放到一个静态变量中
private static string connStr =
    WebConfigurationManager.ConnectionStrings["SqlServer2005ConnectionString"].ConnectionString;
```

数据访问层内的很多方法是以静态的方式存在的，调用方只需使用类名即可快速地访问，无须创建对应的对象实例，因此内部的成员变量和成员方法基本采用类变量和类方法的方式定义。

（6）利用 ADO.NET 对数据查询接口进行封装，用于返回多条记录集。在数据库的操作过程中经常要进行查询，并且满足查询条件的会有多条，为了查询的便利，把此类数据查询封装成一个方法，以方便快速地实现查询。

```
    /// <summary>
    /// 用于执行普通查询，并以 DataTable 离线的方式返回所查询的结果集
```

```
/// </summary>
/// <param name="cmdText">查询的 SQL 语句或存储过程名称</param>
/// <param name="cmdType">Command 的操作类型</param>
/// <param name="cmdParameters">操作所使用到的参数集合</param>
/// <returns></returns>
public static DataTable ExecuteReader(string cmdText, CommandType cmdType,
                                      List<SqlParameter> cmdParameters)
{
    //创建连接对象
    SqlConnection conn = new SqlConnection();
    //设置连接字符串
    conn.ConnectionString = connStr;
    //打开连接
    conn.Open();
    //创建命令对象
    SqlCommand comm = new SqlCommand();
    //设置命令对象的数据源连接对象
    //进行的操作（SQL 语句、数据表名、存储过程）
    //以及操作的类型
    comm.Connection = conn;
    comm.CommandText = cmdText;
    comm.CommandType = cmdType;
    //添加操作所用的参数
    if (cmdParameters != null)
    {
        foreach (SqlParameter para in cmdParameters)
            comm.Parameters.Add(para);
    }
    //创建离线内存表
    DataTable dataTable = new DataTable();
    //创建 Adapter 对象，并利用查询命令的返回结果填充离线内存表
    SqlDataAdapter ad = new SqlDataAdapter();
    ad.SelectCommand = comm;
    ad.Fill(dataTable);
    //释放相关资源
    comm.Parameters.Clear();
    comm.Dispose();
    conn.Close();
    conn.Dispose();
    //返回结果集
    return dataTable;
}
```

（7）对 ADO.NET 的数据查询接口进行封装，用于根据主键获取单条记录。在数据库操作中，经常要利用表的主键进行数据查询，此种情况下只会返回一条记录。为便于对查询结果的操作，将此类查询封装为一个方法，查询的返回结果是一个 DataRow 对象，可以方便地通过此对象对查询的结果进行访问。

```
/// <summary>
/// 用于执行按照主键进行的查询，并以 DataRow 离线的方式返回所查询的单条记录
/// 主要用于根据主键获取单条记录和判断对应的主键是否已被占用
/// </summary>
/// <param name="cmdText"></param>
/// <param name="cmdType"></param>
/// <param name="isExist">用于判断是否有结果</param>
/// <param name="cmdParameters"></param>
/// <returns></returns>
public static DataRow ExecuteReader(string cmdText, CommandType cmdType,
                                    out Boolean isExist, List<SqlParameter> cmdParameters)
{
    SqlConnection conn = new SqlConnection();
    conn.ConnectionString = connStr;
    conn.Open();
    SqlCommand comm = new SqlCommand();
    comm.Connection = conn;
    comm.CommandText = cmdText;
    comm.CommandType = cmdType;
    if (cmdParameters != null)
    {
        foreach (SqlParameter para in cmdParameters)
            comm.Parameters.Add(para);
    }
    DataTable dataTable = new DataTable();
    SqlDataAdapter ad = new SqlDataAdapter();
    ad.SelectCommand = comm;
    ad.Fill(dataTable);
    comm.Parameters.Clear();
    comm.Dispose();
    conn.Close();
    conn.Dispose();
    //利用 DataTable 内的记录条数判断是否查询出一条记录，有且只能有一条
    if (dataTable.Rows.Count == 1)
    {
        isExist = true;
        return dataTable.Rows[0];
```

```
        }
        else
        {   isExist = false;     }
        return null;
    }
```

（8）对 ADO.NET 的数据编辑接口进行封装，用于普通的无事务要求单条编辑操作。在数据库操作中，经常要对数据进行增加、修改、删除等操作，将此操作封装为一个方法以便调用。

```
    /// <summary>
    /// 用于普通的无事务要求单条编辑操作
    /// </summary>
    /// <param name="cmdText"></param>
    /// <param name="cmdType"></param>
    /// <param name="cmdParameters"></param>
    /// <returns></returns>
    public static Boolean ExecuteNonQuery(string cmdText, CommandType cmdType,
                                    List<SqlParameter> cmdParameters)
    {
        SqlConnection conn = new SqlConnection();
        conn.ConnectionString = connStr;
        conn.Open();
        SqlCommand comm = new SqlCommand();
        comm.Connection = conn;
        comm.CommandText = cmdText;
        comm.CommandType = cmdType;
        if (cmdParameters != null)
        {
            foreach (SqlParameter para in cmdParameters)
                comm.Parameters.Add(para);
        }
        //执行数据编辑操作
        try
        { comm.ExecuteNonQuery();  }
        catch (SqlException ex)
        { return false;   }
        finally
        {
            comm.Parameters.Clear();
            comm.Dispose();
            conn.Close();
```

```
            conn.Dispose();
        }
        return true;
    }
```

（9）对 ADO.NET 的数据编辑接口进行封装，用于执行连续的多条操作，并要求多条操作拥有事务控制功能。在数据库操作过程中，会有多条操作一起执行，并且要求几个操作之间满足事务的特性。为便于此类操作的执行，将此类操作封装为一个方法。

```
    //定义事务对象，不同的事务所采用的事务对象不同，因此需要定义为域变量
    //因此使用事务的时候需要实例化 DBHelper 类
    private SqlTransaction transtion = null;
    //把事务封装为属性，方便外部获取使用用于事务的提交和回滚
    public SqlTransaction Transtion
    {
        get {    return transtion;    }
    }
    /// <summary>
    ///  开始事务，创建连接对象和初始化事务对象
    /// </summary>
    public void BeginTransaction()
    {
        SqlConnection conn = new SqlConnection();      //创建连接对象
        conn.ConnectionString = connStr;               //设置连接字符串
        conn.Open();                                   //打开连接
        this.transtion = conn.BeginTransaction();
    }
    /// <summary>
    ///  带事务控制多条连续编辑操作接口
    /// </summary>
    /// <param name="cmdText"></param>
    /// <param name="cmdType"></param>
    /// <param name="cmdParameters"></param>
    /// <returns></returns>
    public void ExecuteNonQueryWithTransaction(string cmdText, CommandType cmdType,
                                        List<SqlParameter> cmdParameters)
    {
        SqlCommand comm = new SqlCommand();       //创建命令对象
        //设置命令对象的数据源连接对象，所用的事务对象，
        //进行的操作（SQL 语句、数据表名、存储过程），以及操作的类型
        comm.Connection = this.transtion.Connection;
        comm.Transaction = this.transtion;
```

```
        comm.CommandText = cmdText;
        comm.CommandType = cmdType;
        //添加操作所用的参数
        if (cmdParameters != null)
        {
            foreach (SqlParameter para in cmdParameters)
                comm.Parameters.Add(para);
        }
        comm.ExecuteNonQuery();                    //执行数据编辑操作
    }
    /// <summary>
    /// 结束事务，释放相关资源
    /// </summary>
    public void EndTransaction()
    {   this.transtion.Dispose();    }
```

（10）对 Command 对象的 ExecuteScala 进行封装，返回查询结果的第一行第一列。在数据库操作过程中，经常会用到查询汇总等操作，将此类操作封装为一个方法以方便快速地调用。

```
    /// <summary>
    /// 封装 ExecuteScalar，用来返回查询结果中的第一行第一列
    /// </summary>
    /// <param name="cmdText">操作语句</param>
    /// <param name="cmdType">操作类型</param>
    /// <param name="cmdParameters">操作参数</param>
    /// <returns></returns>
    public static object ExecuteScalar(string cmdText, CommandType cmdType,
                                       List<SqlParameter> cmdParameters)
    {
        SqlConnection conn = new SqlConnection();
        conn.ConnectionString = connStr;
        conn.Open();
        SqlCommand comm = new SqlCommand();
        comm.Connection = conn;
        comm.CommandText = cmdText;
        comm.CommandType = cmdType;
        if (cmdParameters != null)
        {
            foreach (SqlParameter para in cmdParameters)
                comm.Parameters.Add(para);
        }
```

```
        //执行 ExecuteScalar 方法
        object val = comm.ExecuteScalar();
        comm.Dispose();
        conn.Close();
        conn.Dispose();
        return val;
    }
```

（11）为创建 SqlParamater 参数封装一个方法。为了便于快速地创建 SqlCommand 操作所使用的参数对象，将创建参数对象封装为一个方法，以方便快速地调用。

```
    /// <summary>
    ///  创建参数对象
    /// </summary>
    /// <param name="paraName">参数名称</param>
    /// <param name="paraType">参数类型</param>
    /// <param name="paraValue">参数的值</param>
    /// <returns></returns>
    public static SqlParameter CreateParameters(string paraName, DbType paraType, string paraValue)
    {
        SqlParameter para = new SqlParameter();
        para.ParameterName = paraName;
        para.DbType = paraType;
        para.Value = paraValue;
        return para;
    }
```

第二部分　知识点链接

L3.1　系统数据库实现

L1. SQL语句

结构化查询语言 SQL（Structured Query Language）是用于关系数据库操作的标准语言。主要由 3 部分组成：数据定义语言，数据操作语言，数据控制语言。现就本系统所用到的 SQL 语句进行简要的介绍。

1. 数据查询语句SELECT

SELECT 查询是 SQL 语言的核心，功能强大，和各类 SQL 子句结合，可完成复杂的查询操作。在数据库应用中，最常用的操作是查询，查询还是数据库的其他操作（如统计、插入、删除及修改）的基础。

SELECT 语句很复杂，主要的子句如下：

```
SELECT [DISTINCT] [别名.]字段名或表达式 [AS 列标题]
                                        /* 指定要选择的列或行及其限定 */
FROM   table_sources                    /* FROM 子句，指定表或视图 */
[ WHERE   search_condition ]             /* WHERE 子句，指定查询条件 */
[ GROUP BY group_by_expression ]        /* GROUP BY 子句，指定分组表达式 */
[ ORDER BY order_expression [ ASC | DESC ]]
```

其中，SELECT 和 FROM 子句是不可缺少的。

① SELECT 子句指出查询结果中显示的字段名，以及字段名和函数组成的表达式等。可用 DISTINCT 去除重复的记录行；AS 列标题指定查询结果显示的列标题。若要显示表中所有字段，可用通配符“*”代替字段名列表。

② FROM 字句用来指定所查询的表或视图，既可以是单个表或视图，也可以是多个表或视图，多个表或视图之间采用“,”分割。

③ WHERE 子句定义了查询条件。WHERE 子句必须紧跟 FROM 子句之后，其基本格式为：

```
WHERE <search_condition>
```

其中的 search_condition 为查询条件，常用格式为：

```
{ [ NOT ] <precdicate> | (<search_condition> ) }
     [ { AND | OR } [ NOT ] { <predicate> | (<search_condition>) } ]
} [ ,…n ]
```

其中 predicate 为判定运算，结果为 TRUE、FALSE 或 UNKNOWN，格式为：

```
{ expression { = | < | <= | > | >= | <> | != | !< | !> } expression          /* 比较运算 */
| string_expression [ NOT ] LIKE string_expression [ ESCAPE 'escape_character'    /* 字符串模式匹配 */
  | expression [ NOT ] BETWEEN expression AND expression                    /* 指定范围 */
  | expression IS [ NOT ] NULL                                              /* 是否空值判断 */
  | expression [ NOT ] IN ( subquery | expression [,…n] )                   /* IN 子句 */
  | expression { =|<|<=|>|>=|<>|!=|!<|!> } { ALL|SOME|ANY } ( subquery )    /* 比较子查询 */
  | EXIST ( subquery )                                                      /* EXIST 子查询 */
}
```

从查询条件的构成可以看出，可以将多个判定运算的结果通过逻辑运算符再组成更为复杂的查询条件。判定运算包括比较运算、模式匹配、范围比较、空值比较和子查询等。

在 SQL 中，返回逻辑值（TRUE 或 FALSE）的运算符或关键字都可称为谓词。

GROUP BY 子句和 ORDER BY 子句分别对查询结果分组和排序。

下面用示例说明 SQL 语句的使用，对 Student 数据库进行各种查询。

（1）查询 Student 数据库的 students 表中各个同学的姓名和总学分。

```
USE Student
SELECT name,totalscore FROM students
```

（2）查询表中所有记录。查询 students 表中各个同学的所有信息。

```
SELECT * FROM students
```

（3）条件查询。查询 students 表中总学分大于等于 120 的同学的情况。

```
SELECT * FROM students
        WHERE totalscore >= 120
```

（4）多重条件查询。查询 students 表中所在系为“计算机”且总学分大于等于 120 的同学的情况。

```
SELECT * FROM students WHERE totalscore >= 120 AND major='计算机'
```

（5）使用 LIKE 谓词进行模式匹配。查询 students 表中姓“王”且单名的学生情况。

```
SELECT * FROM students
      WHERE name LIKE '王_'
```

（6）用 BETWEEN…AND 指定查询范围。查询 students 表中不在 1979 年出生的学生情况。

```
SELECT * FROM students
      WHERE birthday NOT BETWEEN '1979-1-1' and '1979-12-31'
```

（7）空值比较。查询总学分尚不定的学生情况。

```
SELECT * FROM students
   WHERE totalscore IS NULL
```

（8）自然连接查询。查找计算机系学生姓名及其“C 程序设计”课程的考试分数情况。

```
SELECT name,grade
      FROM students, courses,grades,
      WHERE department = '计算机' AND student.studentid = grades.studentid
AND courses.courseid = grades.coursesid
```

（9）IN 子查询。查找选修了课程号为 101 的课程的学生的情况。

```
SELECT * FROM students
                    WHERE studentid IN
                          ( SELECT studentid FROM courses WHERE courseid = '101' )
```

在执行包含子查询的SELECT 语句时，系统先执行子查询，产生一个结果表，再执行外查询。本例中，先执行子查询：

```
SELECT studentid FROM courses WHERE courseid = '101'
```

得到一个只含有 studentid 列的结果表，courses 中 courseid 列值为 101 的行在该结果表中都有一行。再执行外查询，若 students 表中某行的 stuentid 列值等于子查询结果表中的任一个值，则该行就被选择到最终结果表中。

（10）比较子查询，这种子查询可以认为是 IN 子查询的扩展，将表达式的值与子查询的结果进行比较运算。查找课程号 206 的成绩不低于课程号 101 的最低成绩的学生的学号。

```
SELECT studentid FROM grades
       WHERE courseid = '206' AND grade !< ANY
              ( SELECT grade FROM grades
                   WHERE courseid = '101'
               )
```

（11）EXISTS 子查询。EXISTS 谓词用于测试子查询的结果是否为空表，若子查询的结果集不为空，则 EXISTS 返回 TRUE，否则返回 FALSE。EXISTS 还可与 NOT 结合使用，即 NOT EXISTS，其返回值与 EXISTS 刚好相反。查找选修 206 号课程的学生姓名。

```
SELECT name FROM students
        WHERE EXISTS
              ( SELECT * FROM grades
                    WHERE studentid = students.studentid AND courseid = '206'
              )
```

（12）查找选修了全部课程的同学的姓名（即查找没有一门功课不选修的学生）。

```
SELECT name FROM students
    WHERE NOT EXISTS
         ( SELECT * FROM courses
                WHERE NOT EXISTS
                ( SELECT * FROM grades
                       WHERE studentid=XS.studentid AND courseid=courses.courseid
                )
         )
```

（13）查询结果分组。将各课程成绩按学号分组。

```
SELECT studentid,grade FROM grades
     GROUP BY studentid
```

（14）查询结果排序。将计算机系的学生按出生时间先后排序。

```
SELECT * FROM students
    WHERE department = '计算机'
      ORDER BY birthday
```

2．数据插入语句INSERT

INSERT 可添加一个或多个记录至表中。

INSERT 有两种语法形式：

```
INSERT INTO target [IN externaldatabase] (fields_list)
     {DEFAULT VALUES|VALUES（DEFAULT|expression_list）
```

或

```
INSERT INTO target [IN externaldatabase] fields_list
     {SELECT…|EXECUTE…}
```

target：欲追加记录的表（Table）或视图（View）的名称。

externaldatabase：外部数据库的路径和名称。

expression_list：需要插入的字段值表达式列表，其个数应与记录的字段个数一致，若指定要插入值的字段 fields_list，则应与 fields_list 的字段个数相一致。

使用第一种形式将一个记录或记录的部分字段插入到表或视图中。第二种形式的 INSERT 语句插入来自 SELECT 语句或来自 EXECUTE 语句执行的存储过程的结果集。

例如，以下语句向 students 表添加一条记录：

```
INSERT INTO students
    VALUES('990206','罗亮', 0 ,'1/30/1980', 1, 150)
```

3．数据更新语句UPDATE

UPDATE 语句用来更新表中的记录。

UPDATE 语句的语法格式如下：

```
UPDATE table_name
SET Field_1=expression_1[,Field_2=expression_2…]
[FROM table1_name|view1_name[,table2_name|view2_name…]]
[WHERE…]
```

Field：需要更新的字段。

Expression：要更新字段的新值表达式。

例如，以下语句将计算机系的学生的总分增加 10：

```
UPDATE students
SET totalscore = totalscore +10
WHERE department = '计算机'
```

4．数据删除语句DELETE

DELETE 用来从一个或多个表中删除记录。

DELETE 语句的语法格式如下：

```
DELETE FROM table_names
[WHERE…]
```

例如，以下语句从 students 表中删除姓名为“罗亮”的记录：

```
DELETE FROM students
    WHERE name = '罗亮'
```

5．存储过程

存储过程（Stored Procedure）是存放于数据库中的子程序，它在服务器端运行，是由一系列 SQL 语句和控制语句组成的数据处理过程。使用存储过程的优点是：

① 更快的执行速度。

② 降低网络流量。

③ 资源共享和安全操作。

④ 灵活的编程及维护方式。

因存储过程具有执行速度快、功能封装和安全操作等优点，所以应用开发中存储过程得到了广泛应用。

在 SQL Server 2005 中创建存储过程的 SQL 命令是 Create Procedure，其语法格式如下：

```
CREATE   PROC[EDURE]   procedure_name                    /* 定义存储过程名 */
[ { @parameter data_type }                               /* 定义参数的类型 */
[ VARYING ] [ = default ] [ OUTPUT ] ]                   /* 定义参数的属性 */
```

```
[ ,...n1]
[ WITH  { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION }]
                                                    /* 定义存储过程的处理方式 */
[ FOR REPLICATION ]
AS
BEGIN
sql_statement   [ ...n2 ]                           /* 执行的操作 */
END
```

其中，procedure_name 是定义的存储过程名，其他参数的含义如下：

@parameter 为存储过程的形参，@符号作为第一个字符来指定参数名称。参数名必须符合标识符规则，创建存储过程时，可声明一个或多个参数，执行存储过程时应提供相应的实参，除非定义了该参数的默认值，默认参数值只能为常量。形参局部于该存储过程。data_type 用于指定形参数据类型，形参可为 SQL Server 支持的任何类型。

default 指定存储过程输入参数的默认值，默认值必须是常量或 NULL，默认值中可以包含通配符（%、_、[]和[^]），如果定义了默认值，执行存储过程时根据情况可不提供实参。

关键字 OUTPUT 用于指定参数从存储过程返回信息。

n1 表示可为存储过程指定若干参数。

sql_statements 是 SQL 语句，用于描述所定义的存储过程所要执行的操作。

n2 表示可以有多个 SQL 语句，多个 SQL 语句一起组成存储过程需要执行的一系列操作。

例如：

```
CREATE PROCEDURE InsertStudent
    @CN nchar(10),
    @CS int,
    @Age int
AS
BEGIN
    Insert INTO Course (courseName, courseScore) Values (@CN,@CS)
END
```

6．触发器

触发器是一种特殊类型的存储过程，当使用一种或多种数据修改操作（UPDATE、INSERT 或 DELETE）来修改指定表中的数据时运行。

触发器可以查询其他表，而且可以包含复杂的 SQL 语句。它们主要用于强制服从复杂的业务规则或要求。例如，可以根据客户当前的账户状态，控制是否允许插入新订单。

触发器也可用于强制引用完整性，以便在多个表中添加、更新或删除行时，保留在这些表之间所定义的关系。使用触发器的优点如下：

① 触发器是自动的。它们可在对表的数据做了任何修改（如手工输入或者应用程序采取的操作）之后立即被激活。

② 触发器可以通过数据库中的相关表进行级联更改。例如，可以在 titles 表的 title_id

列上编写一个删除触发器，以触发在其他表中删除各匹配行的操作。该触发器用 title_id 列作为唯一键，在 titleauthor、sales 和 roysched 表中对各匹配行进行定位。

③ 触发器可以强制限制，这些限制比用 CHECK 约束所定义的更复杂。与 CHECK 约束不同的是，触发器可以引用其他表中的列。

在触发器中使用了两个特殊的表：deleted 表和 inserted 表。它们都是概念上的表。这些表在结构上类似于定义触发器的表（也就是在其中尝试用户操作的表）；用于保存用户操作可能更改的行的旧值或新值。

在 SQL Server 2005 中创建触发器的 SQL 命令是 CREATE TRIGGER，其语法格式如下：

```
CREATE TRIGGER trigger_name
ON { table | view }
{ [AFTER | INSTEAD OF] } { [ INSERT ] [ UPDATE ] [ DELETE ] }
AS
BEGIN
        sql_statement [ ...n ]
END
```

其中，trigger_name 是定义的存储过程名，sql_statement 是一组 SQL 语句，用于描述所定义的存储过程要执行的操作。其他参数的含义如下：

table | view 是在其上执行触发器的表或视图，有时称为触发器表或触发器视图。可以选择是否指定表或视图的所有者名称。

AFTER 指定触发器只有在触发 SQL 语句中指定的所有操作都已成功执行后才激发。所有的引用级联操作和约束检查也必须成功完成后，才能执行此触发器。

INSTEAD OF 是“代替”的意思，也就是原本直接由 SQL Server 执行的添加、修改、删除操作，全部替换成由 INSTEAD OF 触发器执行。

sql_statement 是触发器所要执行的操作。

n 是多条 SQL 语句共同组成触发器所要执行的操作。

L3.2　构建数据访问层

L1. ADO.NET简介

ADO.NET 是 ActiveX Data Objects for the .NET Framework 的缩写，它是为.NET 框架创建的，提供了对 Microsoft SQL Server、Oracle 等数据源以及通过 OLE DB 和 XML 公开的数据源的一致访问。应用程序可以使用 ADO.NET 连接到这些数据源，并检索、操作和更新数据。

ADO.NET 是一个全新的数据访问模型，该模型从 ADO 发展而来，但它不只是对 ADO 的改进，而是采用了一种全新的技术。

ADO.NET 既能在与数据源连接的环境下工作，又能在断开与数据源连接的条件下工作。特别是后者，非常适合于网络应用的需要。因为在网络环境下，保持与数据源连接，不符合网站的要求，不仅效率低，付出的代价高，而且常常会引发由于多个用户同时访问时带来的冲突。

ADO.NET 针对客户端和数据库的连接提供了两种环境：连接环境（Connected）与非连接环境（Disconnected）。

1．连接环境

对于过去的大部分数据访问接口而言，唯一可用的环境就是连接环境。在有连接的环境中，应用程序与数据库保持持续的连接。

2．非连接环境

随着 Internet 的出现，无连接的环境日益普及。同时随着手持设备的增加，当与服务器或者数据库断开连接时，仍可以通过计算机或手持设备使用应用程序。

在非连接环境中，应用程序首先从数据源中把数据读入到内存中，在内存中通过数据集（ADO.NET 提供的离线数据库对象模型）构建出一个内存数据库，在数据集中的数据可以被独立地复制和更改，需要时可以与数据源中的数据合并。

在许多情况下，并不是在完全有连接或完全无连接的环境下工作，而是在两种方式混合环境下工作。

L2. ADO.NET核心组件

在 ADO.NET 中数据提供程序（Provider）与数据集（DataSet）是两个非常重要而又相互关联的核心组件。它们之间的关系如图 3.7 所示。

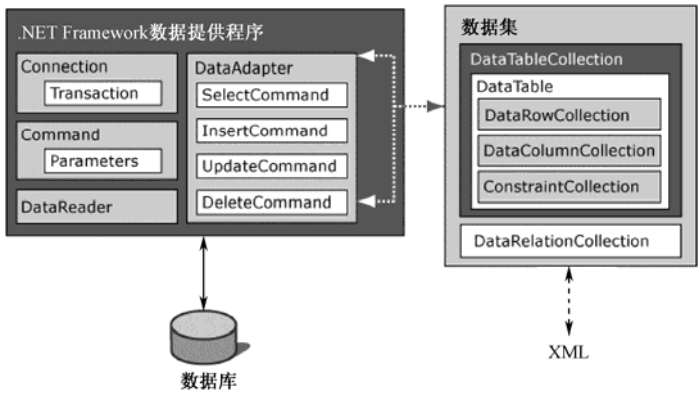


图 3.7　ADO.NET 核心组件

数据集是实现 ADO.NET 断开式连接的核心，从数据源读取的数据先缓存到数据集中，然后被程序或控件调用。

数据提供程序用于建立数据源与数据集之间的联系，它能连接各种类型的数据，并能按要求将数据源中的数据提供给数据集，或者从数据集向数据源返回编辑后的数据。

1. 数据提供程序

ADO.NET 数据提供程序提供了连接到数据源、检索和编辑数据等功能，并且针对不同的数据源提供了不同的数据提供程序，连接到具体的数据库的时候应该选择相应的数据提供程序。

Microsoft .NET Framework 中包含以下 4 个.NET 数据提供程序：

① SQL Server .NET 数据提供程序：用于专门连接 SQL Server 的数据库（较老的产品不支持），并针对 SQL Server 数据库提供了额外的功能。该数据提供程序包含在 System.Data.SqlClient 命名空间中。

② Oracle .NET 数据提供程序：用于专门连接 Oracle 的数据库（较老的产品不支持），并针对 Oracle 数据库提供了额外的功能。该数据提供程序包含在 System.Data.OracleClient 命名空间中。

③ OLE DB .NET 数据提供程序：用于连接 OLE DB 数据库，以兼容早期的数据库产品。该数据提供程序包含在 System.Data.OleDb 命名空间中。

④ ODBC .NET 数据提供程序：用于连接 ODBC 数据源，通过 ODBC 数据提供程序可以连接所有 ODBC 数据提供程序所支持的数据源。该数据提供程序包含在 System.Data.Odbc 命名空间中。

每个.NET Framework 数据提供程序包含 4 种核心对象，其名称及作用如下：

（1）Connection

Connection 建立与特定数据源的连接。在进行数据库操作之前，首先要建立对数据库的连接。所有 Connection 对象的基类均为 DbConnection 类。Connection 类中最重要的属性是 ConnectionString，该属性用来指定建立数据库连接所需要的连接字符串，其中包括如下几项：服务器名称、数据源信息及其他登录信息。ConnectionString 的主要参数有：

① Data Source：设置连接的数据库服务器名。

② Initial Catalog：设置连接的数据库名称。

③ User ID：登录 SQL Server 的账号。

④ Password：登录 SQL Server 的密码。

⑤ Connection Timeout：设置 SqlConnection 对象连接 SQL 数据库服务器的超时时间，单位为秒，若在所设置的时间内无法连接数据库，则返回失败。默认为 15 秒。

以 SQL Server 数据库的连接对象为例，类名为 SqlConnection，其创建的语句是：

```
SqlConnection conn = new SqlConnection();
```

设置 ConnectionString 属性的语句是：

```
conn.ConnectionString =
"Data Source=MySQLServer;                                    // 服务器名
user id=sa;password=123456;                                  // 安全信息
Initial catalog=Northwind; Integrated Security=False";       // 数据库名及其他参数
```

要更好地掌握 Connection 对象的使用，还需要了解如表 3.7 所示的几个常用方法。

表 3.7　Connection 对象的常用方法

方　法	说　明
Open()	打开与数据库的连接。注意 ConnectionString 属性只对连接属性进行了设置，并不打开与数据库的连接，必须使用 Open()方法打开连接
Close()	关闭数据库连接
Dispose()	调用 Close()方法关闭与数据库的连接，并释放所占用的系统资源

注意：在完成数据库操作后，及时关闭连接是很必要的，因为大多数数据源只支持有限数目的连接，而打开的连接占用了宝贵的系统资源。

（2）Command

Command 是对数据源操作命令的封装，所有 Command 对象的基类均为 DbCommand 类。对于数据库来说，这些命令既可以是内联的 SQL 语句，也可以是数据库的存储过程。由 Command 生成的对象建立在连接的基础上，对连接的数据源指定相应的操作。

每个.NET Framework 数据提供程序包括一个 Command 对象：OLEDB .NET Framework 数据提供程序包括 OleDbCommand 对象；SQL Server .NET Framework 数据提供程序包括 SqlCommand 对象；ODBC .NET Framework 数据提供程序包括 ODBCCommand 对象；Oracle .NET Framework 数据提供程序包括 OracleCommand 对象。

以下代码示例演示如何创建 SqlCommand 对象，以便从 SQL Server 中的 Categories 示例数据库返回类别列表。

```
string sql = "SELECT CategoryID, CategoryName FROM Categories";
SqlCommand command1 = new SqlCommand(sql, sqlConnection1);
```

参数 sql 为需执行的 SQL 命令，上述语句将生成一个命令对象 command1，对由 sqlConnection1 连接的数据源指定检索（SELECT）操作。这两个参数在创建 Command 对象时也可以省略不写，而在创建了 Command 对象后，通过设置 Command 对象的 CommandText 和 Connection 等属性来指定。

Command 对象的主要属性和方法分别列于表 3.8 和表 3.9 中。

表 3.8　Command 对象的常用属性

属　性	说　明
CommandText	取得或设置要对数据源执行的 SQL 命令、存储过程或数据表名
CommandTimeout	获取或设置 Command 对象的超时时间，单位为秒，为 0 表示不限制。默认为 30 秒，即若在这个时间之内 Command 对象无法执行 SQL 命令，则返回失败
CommandType	获取或设置命令类别，可取的值：StoredProcedure、TableDirect、Text，代表的含义分别为：存储过程、数据表名和 SQL 语句，默认为 Text。数字、属性的值为 CommandType.StoredProcedure、CommandType.Text 等
Connection	获取或设置 Command 对象所使用的数据连接属性
Parameters	SQL 命令参数集合

表 3.9　Command 对象的常用方法

方　法	说　明
Cancel()	取消 Comand 对象的执行
CreateParameter	创建 Parameter 对象
ExecuteNonQuery()	执行 CommandText 属性指定的内容，返回数据表被影响行数。只有 Update、Insert 和 Delete 命令会影响的行数。该方法用于执行对数据库的更新操作
ExecuteReader()	执行 CommandText 属性指定的内容，返回 DataReader 对象
ExecuteScalar()	执行 CommandText 属性指定的内容，返回结果表第一行第一列的值。该方法只能执行 Select 命令
ExecuteXmlReader()	执行 CommandText 属性指定的内容，返回 XmlReader 对象。只有 SQL Server 才能用此方法

Command 对象的 CommandType 属性用于设置命令的类别，可以是存储过程、表名或 SQL 语句。当将该属性值设为 CommandType.TableDirect 时，要求 CommandText 的值必须是表名，而不能是 SQL 语句。例如：

```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "students";
cmd.CommandType = CommandType.TableDirect;
cmd.Connection = conn;
```

这段代码执行以后，将返回 students 表中的所有记录。它等价于以下代码：

```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "Select * from students";
cmd.CommandType = CommandType.Text;
cmd.Connection = conn;
```

可见，要实现同样功能，可选的办法有多种。

Command 对象提供 4 个执行 SQL 命令的方法：ExecuteNonQuery()、ExecuteReader()、ExecuteScalar()和 ExecuteXmlReader()，要注意每个方法的特点。常用的是 ExecuteNonQuery()和 ExecuteReader()方法，它们分别用于数据库的更新和查询操作。注意 ExecuteNonQuery()不返回结果集而仅返回受影响的行数，ExecuteReader() 返回 DataReader 对象。下面将讲解如何通过 DataReader 对象访问数据库。

（3）DataReader

使用 DataReader 可以实现对特定数据源中的数据进行高速、只读的数据访问。与数据集（DataSet）不同，DataReader 是一个依赖于连接的对象。也就是说，它只能在与数据源保持连接的状态下工作。所有 DataReader 对象的基类均为 DbDataReader 类。

与 Command 类似，每个 .NET Framework 数据提供程序包括一个 DataReader 对象：OLE DB .NET Framework 数据提供程序包括 OleDbDataReader 对象；SQL Server .NET Framework 数据提供程序包括 SqlDataReader 对象；ODBC .NET Framework 数据提供程序包括 OdbcDataReader 对象；Oracle .NET Framework 数据提供程序包括 OracleDataReader 对象。

使用 DataReader 检索数据首先必须创建 Command 对象的实例，然后通过调用 Command 的 ExecuteReader 方法创建 DataReader，以便从数据源检索行。

以下示例说明如何使用 SqlDataReader，其中 command 代表有效的 SqlCommand 对象。

```
SqlDataReader reader = command.ExecuteReader();
```

创建 DataReader 对象后，就可以使用 Read 方法从查询结果中获取行。通过传递列的名称或序号引用，可以访问返回行的每一列。为了实现最佳性能，DataReader 提供了一系列方法，使得能够访问本机数据类型（GetDateTime、GetDouble、GetGuid、GetInt32 等）的列值。

以下代码示例循环访问一个 DataReader 对象，并从每行中返回两列。

```
if (reader.HasRows)                                    //判断是否有结果返回
   while (reader.Read())                               //依次读取行
```

```
        Console.WriteLine("\t{0}\t{1}", reader.GetInt32(0), reader.GetString(1));
    else
        Console.WriteLine("No rows returned.");
    reader.Close();
```

每次使用完 DataReader 对象后都应调用 Close 方法显式关闭。

DataReader 对象的常见属性和方法分别列于表 3.10、表 3.11 中。

表 3.10　DataReader 对象的常用属性

属　性	说　明
HasRows	判断当前是否有记录可以读取
FieldCount	获取 DataReader 对象包含的记录行数
IsClosed	获取 DataReader 对象的状态，为 True 表示关闭
Item({name,col})	获取或设置表字段值，name 为字段名，col 为列序号，序号从 0 开始。例如：objReader.Item(0)、objReader.Item("name")
ReacordsAffected	获取在执行 Insert、Update 或 Delete 命令后受影响的行数。该属性只有在读取完所有行且 DataReader 对象关闭后才会被指定

表 3.11　DataReader 对象的常用方法

方　法	说　明
Close()	关闭 DataReader 对象
GetBoolean(Col)	获取序号为 Col 的列的值，所获取列的数据类型必须为 Boolean 类型；其他类似的方法还有：GetByte、GetChar、GetDateTime、GetDecimal、GetDouble、GetFloat、GetInt16、GetInt32、GetInt64、GetString 等
GetDataTypeName(Col)	获取序号为 Col 的列的来源数据类型名
GetFieldType(Col)	获取序号为 Col 的列数据类型
GetName(Col)	获取序号为 Col 的列的字段名
GetOrdinal(Name)	获取字段名为 Name 的列的序号
GetValue(Col)	获取序号为 Col 的列的值
GetValues(values)	获取所有字段的值，并将字段值存放在 values 数组中
IsDBNull(Col)	若序号为 Col 的列为空值，则返回 True，否则返回 False
Read()	读取下一条记录，返回布尔值。返回 True 表示有下一条记录，返回 False 表示没有下一条记录

（4）DataAdapter

数据适配器（DataAdapter）利用连接对象（Connection）连接数据源，使用命令对象（Command）规定的操作从数据源中检索出数据送往数据集，或者将数据集中经过编辑后的数据送回数据源。所有 DataAdapter 对象的基类均为 DbDataAdapter 类。

如果所连接的是 SQL Server 数据库，则可以通过将 SqlDataAdapter 与关联的 SqlCommand 和 SqlConnection 对象一起使用，从而提高总体性能。对于支持 OLEDB 的数据源，可以使用 OleDbDataAdapter 及其关联的 OleDbCommand、OleDbConnection 对象。对于支持 ODBC 的数据源，使用 OdbcDataAdapter 及其关联的 OdbcCommand、OdbcConnection 对象。对于

Oracle 数据库，使用 OracleDataAdapter 及其关联的 OracleCommand、OracleConnection 对象。

定义 OleDbDataAdapter 对象的语法格式有 4 种：

```
OleDbDataAdapter 对象名 = new OleDbDataAdapter();
OleDbDataAdapter 对象名 = new OleDbDataAdapter(OleDbCommand 对象);
OleDbDataAdapter 对象名 = new OleDbDataAdapter (SQL 命令，OleDbConnection 对象);
OleDbDataAdapter 对象名 = new OleDbDataAdapter (SQL 命令，OleDbConnection 对象)
```

创建 SqlDataAdapter 对象语法格式与之类似，只要将所有的“OleDb”改为“Sql”即可。创建 DataAdapter 对象的这几种格式，读者可根据需要自行选择使用。

DataAdapter 有一个重要的 Fill 方法，此方法将数据填入数据集，语句如下：

```
dataAdapter1.Fill (dataSet1, "Products");
```

其中，dataAdapter1 代表数据适配器名；dataSet1 代表数据集名；Products 代表数据表名。

当 dataAdapter1 调用 Fill() 方法时，将使用与之相关联的命令组件所指定的 SELECT 语句从数据源中检索行，然后将行中的数据添加到 DataSet 中的 DataTable 对象中，如果 DataTable 对象不存在，则自动创建该对象。

当执行上述 SELECT 语句时，与数据库的连接必须有效，但不需要用语句将连接对象打开。如果调用 Fill()方法之前与数据库的连接已经关闭，则将自动打开它以检索数据，执行完毕后再自动将其关闭。如果调用 Fill()方法之前连接对象已经打开，则检索后继续保持打开状态。

一个数据集中可以放置多张数据表。但是每个数据适配器只能够对应于一张数据表。

2. 数据集

数据集相当于内存中暂存的数据库，不仅可以包括多张数据表，还可以包括数据表之间的关系和约束。允许将不同类型的数据表复制到同一个数据集中（其中某些数据表的数据类型可能需要做一些调整），甚至还允许将数据表与 XML 文档组合到一起协同操作。

数据集从数据源中获取数据以后就断开了与数据源之间的连接。允许在数据集中定义数据约束和表关系，增添、删除和编辑记录，还可以对数据集中的数据进行查询、统计等。当完成各项数据操作以后，还可以将数据集中的最新数据更新到数据源。

数据集的这些特点为满足多层分布式应用的需要跨进了一大步。因为编辑和检索数据都是一些比较繁重的工作，需要跟踪列模式、存储关系数据模型等。如果在连接数据源的条件下完成这些工作，不仅会使总体性能下降，还会影响到可扩展性的问题。

创建数据集对象的语句是：

```
DataSet ds = new DataSet ();
```

或

```
DataSet ds = new DataSet ("数据集名");
```

语句中 ds 代表数据集对象。可以通过调用 DataSet 的两个重载构造函数来创建 DataSet 的实例，并且可以选择指定一个名称参数。如果没有为 DataSet 指定名称，则该名称会设置为“NewDataSet”。

DataSet 对象的常用属性如表 3.12 所示。

表 3.12　DataSet 对象的常用属性

属　性	说　明
CaseSensitive	获取或设置在 DataTable 对象中字符串比较时是否区分字母的大小写。默认为 False
DataSetName	获取或设置 DataSet 对象的名称
EnforceConstraints	获取或设置执行数据更新操作时是否遵循约束。默认为 True
HasErrors	DataSet 对象内的数据表是否存在错误行
Tables	获取数据集的数据表集合（DataTableCollection），DataSet 对象的所有 DataTable 对象都属于 DataTableCollection

DataSet 对象最常用的属性是 Tables，通过该属性，可以获得或设置数据表行、列的值。例如，表达式 DS.Tables["students"].Rows[i][j]表示访问 students 表的第 i 行第 j 列。

DataSet 对象的常用方法有 Clear()和 Copy()，Clear()方法清除 DataSet 对象的数据，删除所有 DataTable 对象；Copy()方法复制 DataSet 对象的结构和数据，返回与本 DataSet 对象具有同样结构和数据的 DataSet 对象。

在数据集中包括以下几种子类：

（1）数据表集合（DataTableCollection）和数据表（DataTable）

DataSet 的所有数据表包含于数据表集合 DataTableCollection 中，通过 DataSet 的 Tables 属性访问 DataTableCollection。DataTableCollection 有以下两个属性：

① Count：DataSet 对象所包含的 DataTable 个数。

② Tables[index,name]：获取 DataTableCollection 中下标为 index 或名称为 name 的数据表。如 DS.Tables[0]表示数据集对象 DS 中的第 1 个数据表，DS.Tables[1]表示第 2 个数据表，依次类推。DS.Tables["students"]表示数据集对象 DS 中名称为“students”的数据表。

DataTableCollection 有以下常用方法：

① Add({table,name})：向 DataTableCollection 中添加数据表。

② Clear()：清除 DataTableCollection 中的所有数据表。

③ CanRemove(table)：判断参数 table 指定的数据表能否从 DataTableCollection 中删除。

④ Contains(name)：判断名为 name 的数据表是否被包含在 DataTableCollection 中。

⑤ IndexOf({table,name})：获取数据表的序号。

⑥ Remove({table,name})：删除指定的数据表。

⑦ RemoveAt(index)：删除下标为 index 的数据表。

DataTableCollection 中每个数据表都是一个 DataTable 对象。

DataTable 表示内存中关系数据的表，可以独立创建和使用，也可以由其他 .NET Framework 对象使用，最常见的情况是作为 DataSet 的成员使用。

可以使用相应的 DataTable 构造函数创建 DataTable 对象。可以通过使用 Add 方法将其添加到 DataTable 对象的 Tables 集合中，将其添加到 DataSet 中。

创建 DataTable 时，不需要为 TableName 属性提供值，可以在其他时间指定该属性，或者将其保留为空。但是，在将一个没有 TableName 值的表添加到 DataSet 中时，该表会得到一个从“Table”（表示 Table0）开始递增的默认名称 Table*N*。

例如，以下示例创建 DataTable 对象的实例，并为其指定名称“Customers”。

```
DataTable workTable = new DataTable("Customers");
```

以下示例创建 DataTable 实例，方法是直接将其添加到 DataSet 的 Tables 集合中。

```
DataSet customers = new DataSet();
DataTable customersTable = customers.Tables.Add("CustomersTable");
```

表 3.13、表 3.14 和表 3.15 分别列出了 DataTable 对象的常用属性、方法和事件。

表 3.13　DataTable 对象的常用属性

属　性	说　明
Columns	获取数据表的所有字段，即 DataColumnCollection 集合
DataSet	获取 DataTable 对象所属的 DataSet 对象
DefaultView	获取与数据表相关的 DataView 对象。DataView 对象可用来显示 DataTable 对象的部分数据。可通过对数据表选择、排序等操作获得 DataView（相当于数据库中的视图）
PrimaryKey	获取或设置数据表的主键
Rows	获取数据表的所有行，即 DataRowCollection 集合
TableName	获取或设置数据表名

表 3.14　DataTable 对象的常用方法

方　法	说　明
Copy()	复制 DataTable 对象的结构和数据，返回与 DataTable 对象具有同样结构和数据的 DataTable 对象
NewRow()	创建一个与当前数据表有相同字段结构的数据行
GetErrors()	获取包含错误的 DataRow 对象数组

表 3.15　DataTable 对象的事件

事　件	说　明
ColumnChanged	当数据行中某字段值发生变化时将触发该事件。该事件参数为 DataColumnChangeEventArgs，可以取的值为：Column（值被改变的字段）；Row（字段值被改变的数据行）
RowChanged	当数据行更新成功时将触发该事件。该事件参数为 DataRowChangeEventArgs，可以取的值为：Action（对数据行进行的更新操作名，包括：Add—加入数据表；Change—修改数据行内容；Commit—数据行的修改已提交；Delete—数据行已被删除；RollBack—数据行的更改被取消）；Row（发生更新操作的数据行）
RowDeleted	数据行被成功删除后将触发该事件。该事件参数为 DataRowDeleteEventArgs，可以取的值与 RowChanged 事件的 DataRowChangeEventArgs 参数相同

（2）数据列集合（DataColumnCollection）和数据列（DataColumn）

数据表中的所有字段都被存放于数据列集合 DataColumnColection 中，通过 DataTable 的 Columns 属性访问 DataColumnCollection。例如，stuTable.Columns[i].Caption 代表 stuTable 数据表的第 i 个字段的标题。DataColumnColection 有以下两个属性：

① Count：数据表所包含的字段个数。

② Columns[index,name]：获取下标为 index 或名称为 name 的字段。例如，DS.Tables[0].

Columns[0]表示数据表 DS.Tables[0]中的第 1 个字段；DS.Tables[0].Columns["studentid"]表示数据表 DS.Tables[0]的字段名为 studentid 的字段。

DataColumnCollection 的方法与 DataTableCollection 类似。

数据表中的每个字段都是一个 DataColumn 对象。

DataColumn 对象定义了表的数据结构。例如，可以用它确定列中的数据类型和大小，还可以对其他属性进行设置。例如，确定列中的数据是否是只读的、是否是主键、是否允许空值等；还可以让列在一个初始值的基础上自动增值，增值的步长也可以自行定义。

获取某列的值需要在数据行的基础上进行。语句如下：

```
string dc = dr.Columns["字段名"].ToString();
```

或

```
string dc = dr.Column[index].ToString();
```

两条语句具有同样的作用。其中 dr 代表引用的数据行，dc 是该行某列的值（用字符串表示），index 代表列（字段）对应的索引值（列的索引值从 0 开始）。

综合前面的语句，要取出数据表（dt）中第 3 条记录中的“姓名”字段，并将该字段的值放入一文本框（textBox1）中，语句可以写成：

```
DataTable dt = ds.Tables["Customers"]                    //从数据集中提取数据表 Customers
DataRow dRow = dt.Rows[2 ];                              //从数据表提取第 3 行记录
string textBox1.Text=dRow["CompanyName"].ToString(); //从行中取出名为 CompanyName 字段的值
```

语句执行的结果是：从 Customers 数据表的第 3 条记录中，取出字段名为 CompanyName 的值，并赋给 textBox1.Text。

表 3.16 列出了 DataColumn 对象的常用属性。

表 3.16　DataColumn 对象的常用属性

属　性	说　明
AllowDBNull	设置该字段可否为空值。默认值为 True
Caption	获取或设置字段标题。若未指定字段标题，则字段标题即为字段名。该属性常与 DataGrid 配合使用
ColumnName	获取或设置字段名
DataType	获取或设置字段的数据类型
DefaultVale	获取或设置新增数据行时，字段的默认值
ReadOnly	获取或设置新增数据行时，字段的值是否可修改。默认值为 False
Table	获取包含该字段的 DataTable 对象

通过 DataColumn 对象的 DataType 属性设置字段数据类型时，不可直接设置数据类型，而要按照以下语法格式：

DataColumn 对象名.DataType = typeof (数据类型)

其中的“数据类型”取值为.NET Framework 数据类型，常用的值如下：

```
System.Boolean—布尔型                    System.Char—字符型
System.DateTime—日期型                   System.Decimal—数值型
System.Double—双精度数据类型             System.Int16—短整数类型
System.Int32—整数类型                    System.Int64—长整数类型
System.Single—单精度数据类型             System.String—字符串类型
```

（3）数据行集合（DataRowCollection）和数据行（DataRow）

数据表的所有行都被存放于数据行集合 DataRowCollection 中，通过 DataTable 的 Rows 属性访问 DataRowCollection。如 stuTable.Rows[i][j]表示访问 stuTable 表的第 i 行、第 j 列数据。DataRowCollection 的属性和方法与 DataColumnCollection 对象类似，不再赘述。

数据表中的每个数据行都是一个 DataRow 对象。

DataRow 对象是给定数据表中的一行数据，或者说是数据表中的一条记录。DataRow 对象的方法提供了对表中数据的插入、删除、更新和查询等功能。提取数据表中的行的语句如下：

```
DataRow dr = dt.Rows[n];
```

其中，DataRow 代表数据行类；dr 是数据行对象；dt 代表数据表对象; n 代表行的序号（序号从 0 开始）。

DataRow 对象的属性主要有：

① Rows[index,columnName]：获取或设置指定字段的值。

② Table：获取包含该数据行的 DataTable 对象。

DataRow 对象的方法主要有：

① AcceptChanges()：将所有变动过的数据行更新到 DataRowCollection。

② Delete()：删除数据行。

③ IsNull({colName,index,Column 对象名})：判断指定列或 Column 对象是否为空值。

L3. 数据访问层的使用

数据访问层主要针对常用的操作进行了封装，并没有把 ADO.NET 的所有功能包含在其中，在大型的项目中可以针对项目的需求进行扩展。

本数据访问层主要对外提供了如下的几种使用场景：

① 用于执行查询操作。

② 用于执行修改操作（无事务要求）。

③ 用户执行修改操作（有事务要求）。

④ 用于查询统计信息。

下面就每种场景给出一个简单的使用实例（所有的代码都以表 Teacher 为例）。

（1）用于执行查询操作。这种场景中主要用到如下几个方法。

```
public static DataTable ExecuteReader(string cmdText, CommandType cmdType, List<SqlParameter>
        cmdParameters)
public static DataRow ExecuteReader(string cmdText, CommandType cmdType, out Boolean isExist,
        List<SqlParameter> cmdParameters)
public static SqlParameter CreateParameters(string paraName, DbType paraType, string paraValue)
```

【例 3.1】 查询所有教师的信息（上课教师，不包括管理员），则代码可以如下编写。

```
string sql = "select * from Teacher where teaType='1'";                    //定义操作的 sql 语句
//执行查询，把查询的结果放在 DataTable 对象中
DataTable result = commDBHelper.ExecuteReader(sql, CommandType.Text, null);
```

【例 3.2】 查询出教师编号为指定编号的老师。

```
//利用一个变量存储所要查询的教工编号
string inputText = "000001";
//定义操作的 sql 语句
string sql = "select * from Teacher where teaID=@teaID";
//创建参数集合
List<SqlParameter> para = new List<SqlParameter>();
//往参数集合中添加 sql 的参数信息
para.Add(commDBHelper.CreateParameters("@teaID", DbType.Int32, inputText));
//定义一个参数用来判断是否查询到
Boolean isExist = false;
//把查询的结果放在 DataRow 对象中
DataRow row = commDBHelper.ExecuteReader(sql, CommandType.Text, out isExist, para) ;
if (isExist)
{
    //执行有查询结果的代码
}
else
{
    //执行无查询结果的代码
}
```

（2）用于执行修改操作（无事务要求）。这种场景中主要用到如下几个方法。

```
public static Boolean ExecuteNonQuery(string cmdText, CommandType cmdType,  List<SqlParameter>
        cmdParameters)
public static SqlParameter CreateParameters(string paraName, DbType paraType, string paraValue)
```

【例 3.3】 插入一位新教师。

```
//定义操作的 sql 语句
string sql = @"insert into Teacher(teaName, teaSex, teaTelphone, teaType, teaPhone, teaEmail, teaPassword)
        values(@teaName, @teaSex, @teaTelphone, @teaType, @teaPhone, @teaEmail, @teaPassword)";
//创建参数集合并往集合内添加 sql 语句所需要的参数信息
List<SqlParameter> para = new List<SqlParameter>();
para.Add(commDBHelper.CreateParameters("@teaName", DbType.String, "李辉"));
para.Add(commDBHelper.CreateParameters("@teaSex", DbType.String, "F"));
para.Add(commDBHelper.CreateParameters("@teaTelphone", DbType.String, "13611591202"));
para.Add(commDBHelper.CreateParameters("@teaType", DbType.String, "1"));
para.Add(commDBHelper.CreateParameters("@teaPhone", DbType.String, "025-51667578"));
para.Add(commDBHelper.CreateParameters("@teaEmail", DbType.String, "lihui@126.com"));
para.Add(commDBHelper.CreateParameters("@teaPassword", DbType.String, "lihui@126.com"));
//执行插入操作
commDBHelper.ExecuteNonQuery(sql, CommandType.Text, para);
```

（3）用户执行修改操作（有事务要求）。这种场景中主要用到如下几个方法。

```
public void BeginTransaction()
public void ExecuteNonQueryWithTransaction(string cmdText, CommandType cmdType,List<SqlParameter>
    cmdParameters)
public void EndTransaction()
public static SqlParameter CreateParameters(string paraName, DbType paraType, string paraValue)
public SqlTransaction Transtion
```

【例 3.4】　一次插入多位教师，要求事务支持。

```
//实例化数据库工具类
commDBHelper operation = new commDBHelper();
//开始事务操作
operation.BeginTransaction();
try
{
    //定义操作的 sql 语句
        string sql = @"insert into Teacher(teaName, teaSex, teaTelphone, teaType, teaPhone, teaEmail,
                teaPassword) values(@teaName, @teaSex, @teaTelphone, @teaType, @teaPhone,
                @teaEmail, @teaPassword)";
    //创建参数集合并往集合内添加 sql 语句所需要的参数信息
    List<SqlParameter> para = new List<SqlParameter>();
    para.Add(commDBHelper.CreateParameters("@teaName", DbType.String, "李辉"));
    para.Add(commDBHelper.CreateParameters("@teaSex", DbType.String, "F"));
    para.Add(commDBHelper.CreateParameters("@teaTelphone", DbType.String, "13611591202"));
    para.Add(commDBHelper.CreateParameters("@teaType", DbType.String, "1"));
    para.Add(commDBHelper.CreateParameters("@teaPhone", DbType.String, "025-51667578"));
    para.Add(commDBHelper.CreateParameters("@teaEmail", DbType.String, "lihui@126.com"));
    para.Add(commDBHelper.CreateParameters("@teaPassword", DbType.String, "lihui@126.com"));
    //执行第一位教师的插入操作
    operation.ExecuteNonQueryWithTransaction (sql, CommandType.Text, para);
    //把第一次的参数集合清空，可以继续使用其保存第二次操作的参数信息
    para.Clear();
    para.Add(commDBHelper.CreateParameters("@teaName", DbType.String, "杨娟"));
    para.Add(commDBHelper.CreateParameters("@teaSex", DbType.String, "M"));
    para.Add(commDBHelper.CreateParameters("@teaTelphone", DbType.String, "13611591203"));
    para.Add(commDBHelper.CreateParameters("@teaType", DbType.String, "1"));
    para.Add(commDBHelper.CreateParameters("@teaPhone", DbType.String, "025-51667579"));
    para.Add(commDBHelper.CreateParameters("@teaEmail", DbType.String, "yangjuan@126.com"));
    para.Add(commDBHelper.CreateParameters("@teaPassword", DbType.String, "yangjuan@126.com"));
    //执行第二位老师的插入操作
    operation.ExecuteNonQueryWithTransaction (sql, CommandType.Text, para);
```

```
        //多位的话还可以继续插入
        //提交事务
        operation.Transtion.Commit();
    }
    catch (SqlException ex)
    {    operation.Transtion.Rollback();    }          //如果产生异常，则回滚事务
    finally
    {    operation.EndTransaction();     }             //无论是否发生异常都要结束事务
```

（4）用于查询统计信息。这种场景中主要用到如下几个方法。

```
public static object ExecuteScalar(string cmdText, CommandType cmdType, List<SqlParameter>
      cmdParameters)
public static SqlParameter CreateParameters(string paraName, DbType paraType, string paraValue)
```

【例 3.5】 查询数据库内一共有多少位教师。

```
//定义要查询的 sql
string sql = "select count(teaName) from Teacher";
//指定查询，把结果保存到一个整数中
int count = (int)commDBHelper.ExecuteScalar(sql, CommandType.Text, null);
```

第 4 章　创建学生成绩管理系统主框架

◇ 任务目标

本章主要目标是建立管理员（教务人员）子系统和教师子系统的主框架，设计管理员和教师所使用的主界面，为主界面添加统一的页面功能菜单和页面导航栏，同时为系统内页面添加统一的外观样式定义，为后期系统的开发做一些整体规划和准备工作。完成后的效果如图 4.1 和图 4.2 所示。

图 4.1　管理员子系统主框架

图 4.2　教师子系统主框架

第一部分　应 用 实 践

4.1　创建管理员子系统主界面

1．主框架界面布局设计

在开始创建页面之前，要养成先设计页面布局的习惯。子系统的主界面布局设计如图 4.3 所示。

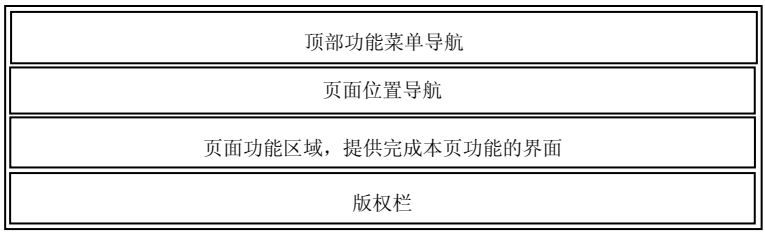


图 4.3　系统主框架布局设计

整个页面和每部分的高度、宽度按照如下规划：

① 页面整体宽度为 700 px；顶部功能菜单导航栏的宽度和页面相同，其高度为 50 px，内容居中；

② 页面位置导航栏宽度和页面相同，高度为 30 px，内容左对齐；

③ 页面功能区域宽度和页面相同，高度随内容的高度自动变化，无须设置高度属性，内容居中；

④ 版权栏宽度和页面相同，高度为 50 px，内容居中。在实际的项目中应该更详细地描述页面的设计规划，便于指导后期页面的开发，也方便指导多人共同开发。

2．创建系统母版页

（1）打开“解决方案资源管理器”窗口，右击 Admin 文件夹，选择“添加新项”，出现“添加新项”对话框，选择“母版页”模板，文件名改为“adminMasterPage.master”，在“语言”下拉框内选择“Visual C#”，选中“将代码放在单独的文件中”复选框，取消“选择母版页”复选框，如图 4.4 所示。

图 4.4　新建母版页

（2）按照相同的方式，给教师添加母版页面，在 Teacher 文件夹下添加一个名为“teacherMasterPage.master”的母版页。

3．利用母版页设计页面模板

这里以设计管理员所用的母版页为例。

（1）打开“解决方案资源管理器”窗口，右击 adminMasterPage.master，选择“视图设计器”，打开 adminMasterPage.master 的设计视图。单击左下角的“拆分”按钮，切换到“拆分”视图。

（2）在“源”视图内，选中 Form 标签内的第一个 div，并设置其 id 属性为“layout”，代码如下。

```
<div id="layout">
        <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
        </asp:ContentPlaceHolder>
</div>
```

（3）在“源”视图内，把光标定位到“ ”的后面，打开“工具箱”窗口，展开“HTML”选项卡，双击“HTML”选项卡内的“**div**”，添加一个 div，设置新增加的 div 的 id 属性为“header”。

（4）在“源”视图内，把光标定位到 id 为“header”的 div 内，展开“工具箱”窗口内的“导航”选项卡，双击“**Menu 控件**”，把其添加到此 div 内。

（5）在“设计”视图内，右击 Menu 控件，选择“属性”，在“属性”窗中内设置 Menu 控件的 Orientation 属性为“Horizontal”，Height 属性为“40 px”，Width 属性为“100%”，单击 Items 属性右侧的按钮，会出现“菜单项编辑器”，单击 图标，在“菜单项编辑器”窗口中添加 9 个根节点，分别设置每个节点的 Text 属性为：教工维护、课程维护、班级维护、学生维护、排课维护、成绩查询、成绩分析、成绩打印、退出系统。

（6）单击 Menu 控件右端的 图标，打开“Menu 任务”菜单，选择“自动套用格式”，弹出“自动套用格式”对话框，选择“简明型”。在 Menu 控件的“属性”窗口中设置 Font 中的 Size 为“1em”。设置结束后，“设计”视图中页面的效果如图 4.5 所示。

图 4.5　Menu 控件的外观

（7）在“源”视图内，把光标定位到 id 为“header”的 div 下方，利用上面的方法在光标位置处插入一个 div，并设置此 div 的 id 属性为“navigate”，光标定位到此 div 内部，双击工具箱的标准选项卡的“SiteMapPath”控件，插入一个站点导航控件“**SiteMapPath**”。

（8）右击“解决方案资源管理器”项目名，选择“添加新项”，在弹出的“添加新项”窗口内选择“**站点地图**”模板，单击“确定”按钮，并按照如下代码修改 **Web.sitemap** 文件。

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="" title="学生成绩管理系统"    description="">
        <siteMapNode url="" title="管理员子系统"    description="" >
            <siteMapNode title="教师信息维护" url="~/Admin/teacher.aspx" />
        </siteMapNode>
    </siteMapNode>
</siteMap>
```

（9）切换到“源”视图，光标定位到 id 为“ContentPlaceHolder1”的控件的下方，利用上面的方法，在光标位置处插入一个 div，并设置 div 的 id 属性为“copyright”。

（10）在“设计”视图内，光标定位到 id 为“copyright”的 div 内，并在 div 内输入“学生成绩”管理系统 2009－2010 版权所有”。母版页中的所有内容设计添加完毕，母版页的效果如图 4.6 所示。

图 4.6　母版页效果图

母版页“源”视图中“<body>”标记块的代码如下所示。

```
<body>
    <form id="form1" runat="server">
    <div id="layout">
        <div id="header">
            <asp:Menu ID="Menu1" runat="server" Orientation="Horizontal"
                BackColor="#E3EAEB" DynamicHorizontalOffset="2" Font-Names="Verdana"
                Font-Size="1em" ForeColor="#666666"
                StaticSubMenuIndent="10px" Height="40px" Width="100%" >
                <StaticSelectedStyle BackColor="#1C5E55" />
                <StaticMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px" />
                <DynamicHoverStyle BackColor="#666666" ForeColor="White" />
                <DynamicMenuStyle BackColor="#E3EAEB" />
                <DynamicSelectedStyle BackColor="#1C5E55" />
                <DynamicMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px" />
                <StaticHoverStyle BackColor="#666666" ForeColor="White" />
                <Items>
                    <asp:MenuItem Text="教工维护" Value="新建项"></asp:MenuItem>
                    <asp:MenuItem Text="课程维护" Value="新建项"></asp:MenuItem>
                    <asp:MenuItem Text="班级维护" Value="新建项"></asp:MenuItem>
                    <asp:MenuItem Text="学生维护" Value="新建项"></asp:MenuItem>
                    <asp:MenuItem Text="排课维护" Value="新建项"></asp:MenuItem>
                    <asp:MenuItem Text="成绩查询" Value="新建项"></asp:MenuItem>
                    <asp:MenuItem Text="成绩分析" Value="新建项"></asp:MenuItem>
                    <asp:MenuItem Text="成绩打印" Value="新建项"></asp:MenuItem>
                    <asp:MenuItem Text="退出系统" Value="退出系统"></asp:MenuItem>
                </Items>
            </asp:Menu>
        </div>
        <div id="navigate">
```

```
                <asp:SiteMapPath ID="SiteMapPath1" runat="server">
                </asp:SiteMapPath>
            </div>
            <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
            </asp:ContentPlaceHolder>
            <div id="copyright">
                学生成绩 管理系统 2009－2010 版权所有</div>
        </div>
        </form>
    </body>
```

（11）打开“解决方案资源管理器”窗口，右击 Admin 文件夹，选择“添加新项”选项，在弹出的“添加新项”对话框中选择“Web 窗体”模板，文件名为“teacher.aspx”，在“语言”下拉框内选择“Visual C#”，选中“将代码放在单独的文件中”和“选择母版页”（套用母版页的 Web 窗体页面称为**内容页**）复选框。单击“确定”按钮，出现“选择母版页”对话框，选择 Admin 文件夹下的“adminMasterPage.master”，单击“确定”按钮。

（12）在“解决方案资源管理器”窗口内，右击“teacher.aspx”，选择“视图设计器”，出现图 4.7 所示的界面。

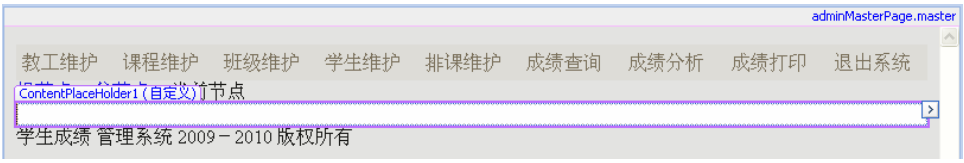


图 4.7　内容页的效果图

功能说明：现在的主要工作是通过母版页为管理者所用的所有页面的顶部添加功能菜单，方便系统功能的导航；并在功能菜单的下部添加一个页面导航功能，能够方便地清楚当前页面所处的位置；并在所有页面尾部添加版权信息。

4.2　美化管理员子系统主界面

1．定义母版页页面样式

（1）右击“主题 1”文件夹，选择“添加新项”选项，在弹出的“添加新项”对话框中选择“**样式表**”模板，名称默认为“StyleSheet.css”。

（2）打开 StyleSheet.css 文件，把光标定位到 body 标签**样式选择符**内，右击鼠标，选择“生成样式”选项，弹出“**修改样式**”对话框，单击左栏类别中的“块”，设置右栏中的 text-align 属性为“center”，使整个页面居中显示，单击“确定”按钮完成样式设计。

（3）右击 Admin 文件夹内的 teacher.aspx 页面，选择视图设计器，进入 teacher.aspx 的“设计”视图，打开“属性”窗口，在顶部下拉框内选择“Document”文档对象，设置 StyleSheetTheme 属性为“主题 1”，页面的外观变化如图 4.8 所示。

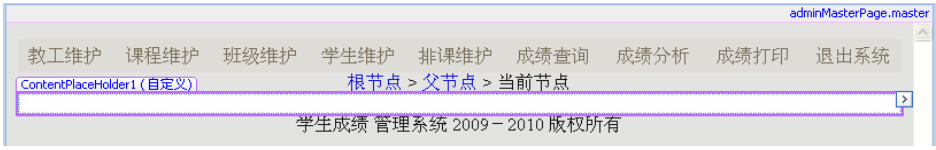


图 4.8　设置页面居中后的页面效果

（4）打开 StyleSheet.css 文件，右击空白处，选择“添加样式规则”选项，弹出“添加样式规则”对话框，在“元素 ID”处输入“layout”，单击“确定”按钮。StyleSheet.css 文件内会添加一个“layout”ID 样式选择器，右击“layout”内部，选择“生成样式”选项，在弹出的“修改样式”对话框中选择左栏类别中的“定位”，设置右栏的 width 属性为“700 px”，单击“确定”按钮完成元素 ID 为“layout”的定位。切换到 teacher.aspx 页面的设计视图，页面外观变化如图 4.9 所示。

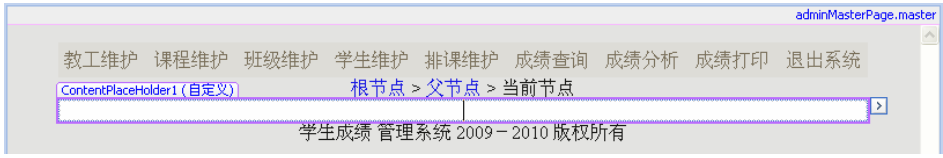


图 4.9　设置页面宽度后的页面效果

（5）打开 StyleSheet.css 文件，右击空白处，选择“添加样式规则”选项，弹出“添加样式规则”对话框，在“元素 ID”处输入“navigate”，单击“确定”按钮。右击“navigate”内部，选择“生成样式”选项，单击左栏中的“定位”类别，设置右栏中的 height 属性为“30 px”；单击左栏中的“块”类别，设置右栏中 line-height 属性为“30 px”，text-align 属性为“left”；单击左栏中的“边框”类别，按照如图 4.10 设置；单击左栏的“方框”类别，按照如图 4.11 设置。

图 4.10　边框设置

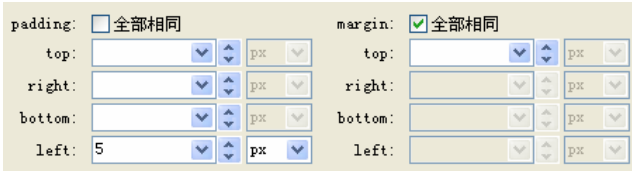


图 4.11　方框设置

切换到 teacher.aspx 页面的设计视图，页面外观变化如图 4.12 所示。

（6）按照同样的方法添加 id 为“copyright”的样式规则，选择“定位”类别，width 属性为“50 px”；选择“块”类别，line-height 属性为“50 px”。切换到 teacher.aspx 页面的设

计视图，页面外观变化如图 4.13 所示。

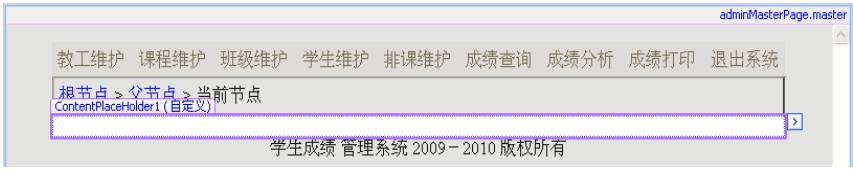


图 4.12　设置导航栏样式后的效果

图 4.13　设置版权后的效果

（7）在 teacher.aspx 页面的空白处，输入“内容页中的内容”，右击空白处，选择“在浏览器中查看”选项，出现如图 4.14 所示的页面效果。

图 4.14　母版页的最终效果

母版页的页面样式已设置完毕，相关的 CSS 代码如下：

```
body
{
    text-align: center;
}
#layout
{
    width: 700px;
}
#navigate
{
    border: thin outset #CCCCCC;
    text-align: left;
    height: 30px;
    line-height: 30px;
    padding-left: 5px;
}
#copyright
```

```
{
    height: 50px;
    line-height: 50px;
}
```

关于内容页所用的样式，在创建具体内容页的时候再进行讲解。

4.3 创建教师子系统主界面

完成管理员子系统的主界面后，建立教师子系统的主界面就比较简单，可以按照如上的方式一步步完成，也可以利用管理员子系统已有的代码快速创建。这里介绍如何快速地创建出教师子系统的主界面。对于学生子系统的主界面的创建，可以作为练习，独自完成。

创建教师子系统主界面步骤如下：

（1）打开“解决方案资源管理器”窗口，右击 teacher 文件夹，选择“添加新项”选项，在 teacher 文件夹内添加一个母版页，命名为“teacherMasterPage.master”（取消“选中模板页”复选框）。

（2）打开“解决方案资源管理器”窗口，右击 adminMasterPage.master，选择“查看标记”选项，在打开的 adminMasterPage.master 的“源”视图中，选中“<body>”标记块内的所有代码，复制并覆盖 teacherMasterPage.master 的“源”视图中的“<body>”标记块内的所有代码。

（3）把 Menu 控件中“<Items>”标记块内的代码改成如下的代码。

```
<Items>
    <asp:MenuItem Text="录入学生成绩" Value="新建项"></asp:MenuItem>
    <asp:MenuItem Text="查询学生成绩" Value="新建项"></asp:MenuItem>
    <asp:MenuItem Text="分析学生成绩" Value="新建项"></asp:MenuItem>
    <asp:MenuItem Text="打印学生成绩单" Value="新建项"></asp:MenuItem>
    <asp:MenuItem Text="解答课程问题" Value="新建项"></asp:MenuItem>
    <asp:MenuItem Text="退出系统" Value="退出系统"></asp:MenuItem>
</Items>
```

（4）打开“解决方案资源管理器”窗口，右击 teacher 文件夹，选择“添加新项”选项，在打开的“添加新项”对话框中选中“Web 窗体”模板，命名为“enterScore.aspx”，语言为“Visual C#”，选中“将代码放在单独的文件中”和“选择母版页”复选框，单击“添加”按钮，在弹出的“选择母版页窗口”对话框中，选中 teacher 文件夹中的 teacherMasterPage.master。

（5）右击 teacher 文件夹内的 enterScore.aspx 页面，选择视图设计器，打开 enterScore.aspx 的“设计”视图。在“属性”窗口中的顶部下拉框内选择“Document”文档对象，设置 StyleSheetTheme 属性为“主题 1”。enterScore.aspx 页面效果如图 4.15 所示。

（6）在空白处输入“内容页中的内容”，右击选择“在浏览器中查看”，出现如图 4.16 所示的页面。

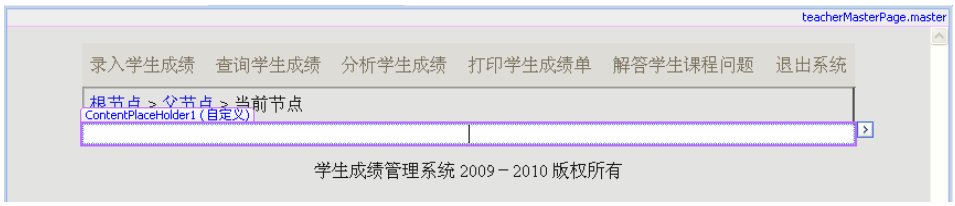


图 4.15　教师子系统中的母版页

图 4.16　教师子系统中的页面主框架

（7）打开“解决方案资源管理器”窗口内的 Web.sitemap 文件，把代码修改如下。

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="" title="学生成绩管理系统"    description="">
        <siteMapNode url="" title="管理员子系统"    description="" >
            <siteMapNode title="教师信息管理" url="~/admin/teacher.aspx" />
          </siteMapNode>
          <siteMapNode url="" title="教师子系统"    description="" >
            <siteMapNode title="录入成绩" url="~/teacher/enterScore.aspx" />
          </siteMapNode>
    </siteMapNode>
</siteMap>
```

（8）在浏览器中查看 enterScore.aspx 页面，设计后的效果如图 4.17 所示，可以看出比图 4.16 多了站点导航。

图 4.17　教师子系统中的页面主框

上面的内容完成了管理员主框架界面的制作和教师主框架界面的制作，并进行相应的外观设置，分别为它们创建一个内容页来查看最终模板页的效果。所用到的相关知识点请参看知识点链接。

第二部分　知识点链接

L4.1　创建管理员子系统主界面

L1. 母版页与内容页

在实际的 Web 项目开发中，大多页面中都有些部分是相同、重复的。如果按照传统的方式，需要对每个页面进行重复设计，不仅浪费时间，日后维护也比较麻烦，需要修改多个页面。母版页是一种很好解决如上问题的手段。母版页是使用比较多的页面模板之一，也是进行专业级 Web 开发必须掌握的基本技术。

1. 母版页和内容页的基本概念

母版页是指其他网页可以将其作为模板来引用的特殊页面。母版页的扩展名为.master。在母版页中，界面分成公用区和可编辑区，公用区是多个内容页中共同的区域，可编辑区是内容页中可以编辑的区域。公用区的设计方法与一般页面相同，可编辑区通过 ContentPlaceHolder 控件为内容页预留出来，ContentPlaceHolder 内部不能进行设计。一个母版页中可以有一个可编辑区，也可以有多个可编辑区。

引用母版页的 Web 窗体页面称为内容页，在内容页中，母版页中 ContentPlaceHolder 控件预留的可编辑区会自动替换为 Content 控件，开发人员只需要在 Content 控件区域中填写内容页中不同的内容即可，在母版页中定义的其他内容将自动出现在引用了该母版页的.aspx 页面中，无须重复设计。

母版页和内容的结构关系如图 4.18 所示。

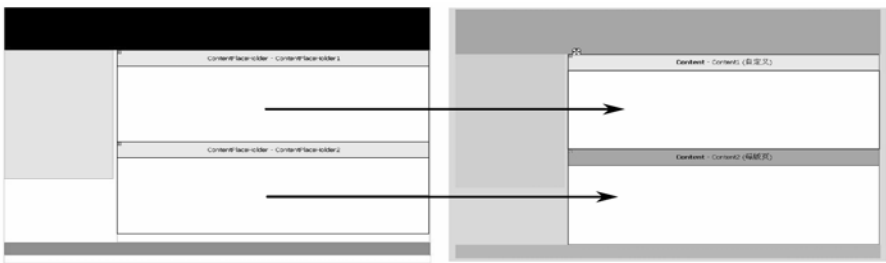


图 4.18　母版页和内容页的结构关系

为了建立起母版页中的 ContentPlaceHolder 控件和内容页中的 Content 控件之间的关系，Content 控件的 ContentPlaceHolderID 属性设置为本 Content 控件所对应的 ContentPlaceHolder 控件的 ID。代码结构如图 4.19 所示。

在运行时，用户和浏览器将按照下面的步骤使用内容页：

① 用户输入内容页的 URL 请求某网页。

② 服务器获取该页后，读取页中的@Page 指令，若该指令引用一个母版页，则读取母版页。如果用户第一次请求这两个页，那么两个页都要进行编译。

③ 服务器将包含的母版页合并到内容页的控件树中。

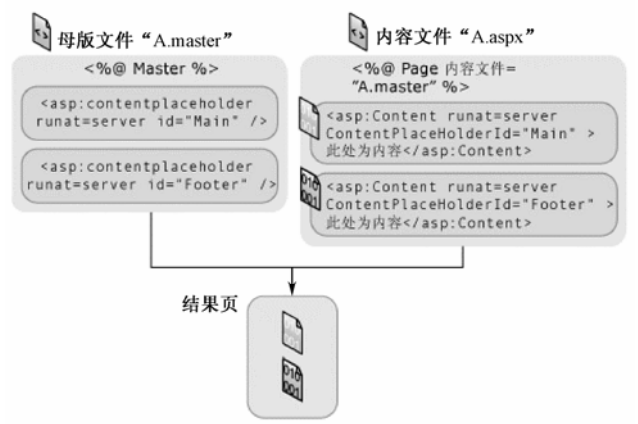


图 4.19　母版页和内容页的代码关系

④ 服务器将页面中各个 Content 控件的内容合并到母版页中相应的 ContentPlace-Holder 控件中。

⑤ 服务器将合并后的内容发送给客户端，客户端在浏览器中呈现合并后的网页。

2. 从内容页中访问母版页

前面只涉及母版页和内容页的页面制作部分，并没有涉及后台代码。但在实际应用中，可能需要通过后台代码从内容页中访问母版页中的控件、属性和方法等。要达到这个目的，必须在母版页中将被访问的属性和方法声明为公共成员 public，如将方法的访问修饰符设置为 public，否则无法从内容页中访问它。从内容页中访问母版页中的控件时，则没有这种限制。

（1）访问母版页中的控件

由于在运行时，母版页与内容页将会合并在一起，从而构成最终的页面，因此内容页的代码可以访问母版页中的控件。具体方法是在内容页后台代码中调用 FindControl 方法获取对母版页中控件的引用。FindControl 方法的原型为：

```
public override Control FindControl(String id)   //id 表示母版页中控件的 ID 名称
```

（2）访问母版页中的公共属性

为了提供对母版页中成员的访问，内容页中提供了 Master 属性来访问母版页对象。但是从内容页中访问母版页的成员，需要在内容页内通过@MasterType 指令来创建对此母版页的强类型引用，该指令的常用形式如下：

```
<%@ MasterType VirtualPath="" %>
```

假如有一个名为 MasterPage1.master 的母版页，其对应的类名为 MasterPage1。在 MasterPage1 类中，声明了一个 TrueName 属性，则可以在内容页代码视图的顶部添加如下代码：

```
<%@ MasterType VirtualPath="MasterPage1.master" %>
```

然后在内容页的后台代码中，就可以通过 Master.TrueName 读取和设置母版页中 TrueName 属性的值。

【例 4.1】 通过一个实例演示如何从内容页访问母版页中的属性和控件。

设计步骤如下：

（1）打开 Visual Studio 2008，新建一个 ASP.NET 网站，网站名称为 Master。

（2）创建一个母版页“MasterPage.master”，设计如图 4.20 所示的界面。

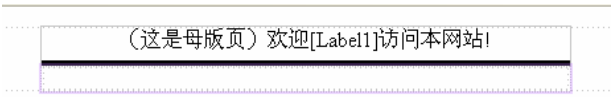


图 4.20　母版页 MasterPage.master

对应的源代码如下：

```
<body style="text-align: center">
    <form id="form1" runat="server">
    <div>
        <div style="width: 400px">
            <div style="border-bottom-color: black;
                    width: 100%; height: 25px; border-bottom-style: solid">
                    （这是母版页）欢迎<asp:Label ID="Label1" runat="server"></asp:Label>
                   访问本网站!</div>
            <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
            </asp:ContentPlaceHolder>
        </div>
    </div>
    </form>
</body>
```

（3）在母版页“MasterPage.master”的后台代码中添加如下的内容：

```
    public string Name
    {
        //Name 的值为 Label 的值
        get { return Label1.Text; }
    }
```

（4）添加一个内容页“content.aspx”，设计如图 4.21 所示的界面。

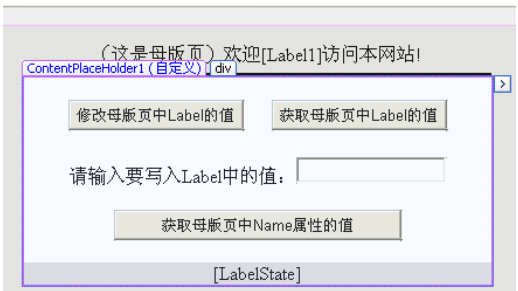


图 4.21　内容页 content.aspx

对应的源代码如下：

```
<div style="width: 400px">
    <br />
    <asp:Button ID="Button1" runat="server" Text="修改母版页中 Label 的值" Width="150px" />
    <asp:Button ID="Button2" runat="server" Text="获取母版页中 Label 的值" Width="150px" />
    <br />
    <br />
    请输入要写入 Label 中的值：<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <br />
    <br />
    <asp:Button ID="Button3" runat="server" Text="获取母版页中 Name 属性的值"  />
    <br />
    <br />
    <asp:Label ID="LabelState" runat="server" BackColor="Gainsboro" Width="100%"></asp:Label>
</div>
```

（5）在内容页的源代码的顶部加入如下代码：

```
<%@ MasterType VirtualPath="~/MasterPage.master" %>
```

（6）分别单击内容页上的按钮，在后台代码中添加如下代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
    //通过 Master.FindControl()方法获取母版页中" Label1"的引用
    Label label1 = (Label)Master.FindControl("Label1");
    label1.Text = TextBox1.Text;
    LabelState.Text = "母版页中 Label1 的值被修改为：" + TextBox1.Text;
}
protected void Button2_Click(object sender, EventArgs e)
{
    //通过 Master.FindControl()方法获取母版页中" Label1"的引用
    Label label1 = (Label)Master.FindControl("Label1");
    LabelState.Text = "母版页中 Label1 的值为：    " + label1.Text;
}
protected void Button3_Click(object sender, EventArgs e)
{
    //通过 Master.Name 访问母版页中的 Name 属性
    LabelState.Text = "母版页中 Name 属性的值为：    " + Master.Name;
}
```

（7）在浏览器中查看内容页，效果如图 4.22 所示。

首先在文本框内输入一个值，然后单击“修改母版页中 Label 的值”按钮，查看页面的变化。再单击“获取母版页中 Label 的值”按钮，查看页面的变化。最后单击“获取母版页中 Name 属性的值”按钮，查看页面的变化。理解整个实例的内部逻辑过程。

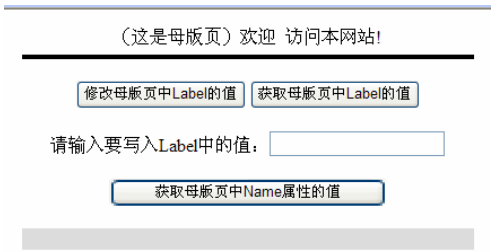


图 4.22　运行结果

L2. 利用Menu控件实现功能菜单导航

Menu 控件主要用于页面中创建系统的功能菜单，让用户可以快速地选择不同功能页面。该控件可以构建出类似于 WinForm 程序的菜单效果。菜单有静态和动态两种显示模式。静态显示模式是指定义的菜单始终完全显示，动态显示模式是指需要用户将鼠标停留在菜单项上时才显示子菜单。

Menu 控件的常用属性如表 4.1 所示。

表 4.1　Menu 控件常用属性

属 性 名	说　明
Orizntation	设置菜单的展开方向。可以取值为 Horizontal 和 Vertical
MaximumDynamicDisplayLevels	设置动态菜单的最大层数，默认为 3

Menu 控件的用法非常灵活，设计者可以利用它定义各种菜单样式，实现类似于 Windows 窗体菜单的功能。下面通过一个例子说明如何利用 Menu 控件实现自定义导航。

【例 4.2】 利用 Menu 控件创建自定义导航。

假定网站包含如下的内容：

班级管理：班级新增（classAdd.aspx），班级查询（classSearch.aspx）

学生管理：学生新增（studentAdd.aspx），学生查询（studentSearch.aspx）

老师管理：老师新增（teacherAdd.aspx），老师查询（teacherSearch.aspx）

具体设计步骤如下：

（1）运行 Visual Studio 2008，新建一个名为“MenuExample”的 ASP.NET Web 应用程序项目，分别在项目内添加需要的网页。

（2）在项目中添加一个名为“MenuExample.aspx”的网页，切换到“设计”视图，向页面中拖放一个 Menu 控件，将 Menu 控件的 Orientation 属性设置为“Horizontal”，以便使其横向排列，单击 Menu 控件右上方的小三角符号，选择“自动套用格式”选项，在弹出的窗口中选择一种喜欢的样式。

（3）单击 Menu 控件右上方的小三角符号，选择“编辑菜单项”选项，在弹出的“菜单项编辑器”对话框中，输入各菜单项，如图 4.23 所示。

（4）在编辑菜单项窗口右侧的属性选项中，利用 NavigateUrl 属性设置各菜单项链接到对应的网页，全部设置完成后，单击“确定”按钮，在浏览器中查看本页，运行效果如图 4.24 所示。

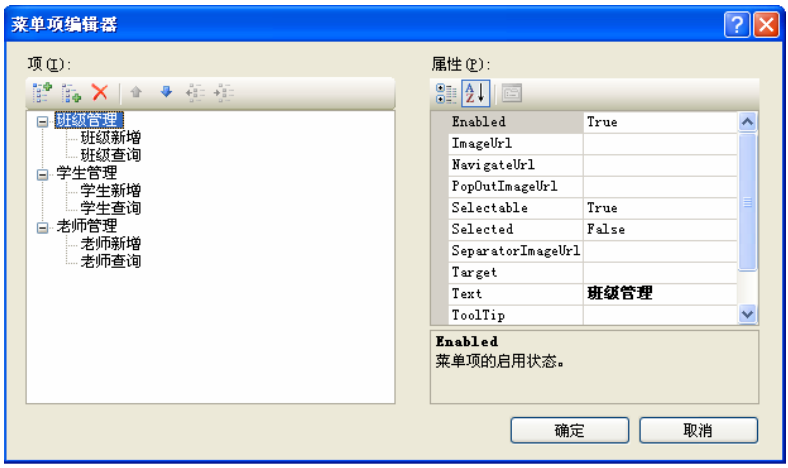


图 4.23　Menu 中菜单项

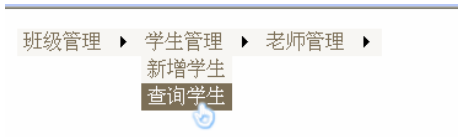


图 4.24　Menu 控件运行效果图

L3. 利用站点地图文件和SiteMapPath实现页面导航

虽然使用超链接或服务器代码可以从一个网页切换到另一个网页，但是，当网站中的页面很多时，页面之间的层次关系将会变得很复杂。ASP.NET 站点导航功能则提供了方便的页面导航。

站点导航主要提供如下功能：

（1）使用站点地图描述站点的逻辑结构，通过 SiteMapPath 可以自动从地图文件中获取文件所在的层次结构，自动生成页面导航栏。

（2）提供导航控件，在页面上显示页面导航菜单。

在 Visual Studio 2008 中，提供的导航控件有 SiteMapPath 控件、Menu 控件和 TreeView 控件。一般情况下，开发人员利用站点地图和 SiteMapPath 控件实现自动导航，利用 Menu 控件或 TreeView 控件实现自定义导航。

站点地图文件 Web.sitemap 是用来描述站点逻辑结构的 XML 文件，该文件的扩展名为.sitemap，站点地图文件必须保存在 Web 应用程序的根目录下才可以。

SiteMapPath 控件以导航路径的方式显示当前页在站点中的位置，定义好站点地图后，只需要将该控件拖放到站点地图中定义过的.aspx 页面上，它就会从.sitemap 地图文件中读取本页所在的节点层次位置，自动显示导航信息，不需要开发者编写任何代码。

注：只有包含在站点地图中的网页才能被 SiteMapPath 控件导航；如果将 SiteMapPath 控件放置在站点地图中未列出的网页中，该控件将不会显示任何信息。

SiteMapPath 控件的常用属性如表 4.2 所示。

表 4.2　SiteMapPath 常见属性

属 性 名	说　　明
CurrentNodeStyle	定义当前节点的外观样式
NodeStyle	定义 SiteMapPath 中所有导航节点的外观样式
PathSeparator	设置导航路径中节点之间的分隔符
PathSeparatorStyle	定义分隔符的样式
RootNodeStyle	定义根节点样式

【例 4.3】通过例子说明如何利用站点地图和 SiteMapPath 控件实现自动导航。

本实例的功能结构如图 4.25 所示。利用 SiteMapPath 控件实现自动导航。

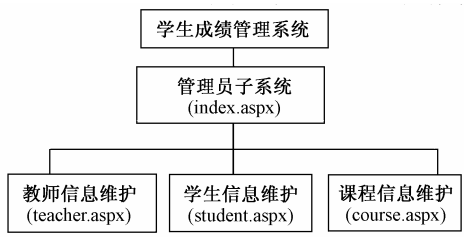


图 4.25　站点结构图

（1）运行 Visual Studio 2008，新建 ASP.NET Web 站点“SiteMapPathExample”。

（2）通过“添加新项”，选择“站点地图”文件类型，如图 4.26 所示，文件名默认为“web.sitemap”。

图 4.26　创建站点地图文件

（3）将 web.sitemap 文件中的内容修改如下，保存文件，完成站点地图设计。

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode title="学生成绩管理系统"    description="">
        <siteMapNode title="管理员子系统" url="index.aspx" description="" >
            <siteMapNode url="teacher.aspx" title="教师信息管理"    description="" />
            <siteMapNode url="student.aspx" title="学生信息管理"    description="" />
```

```
            <siteMapNode url="course.aspx" title="课程信息管理"    description="" />
        </siteMapNode>
    </siteMapNode>
</siteMap>
```

注：站点地图文件中只能有一个根节点，即位于<siteMap>下方的第一个<siteMapNode>元素中，在根节点下可以嵌套任意多个子节点，子节点仍然用 SiteMapNode 定义。如果 SiteMapNode 下边嵌套子节点，则需成对形式出现，子节点位于成对标记内部。如果是叶子节点，则作为单标记形式即可。

SiteMapNode 主要有三个属性：url 用来指定对应页面的相对地址，title 为页面的标题，description 为节点的描述信息。这三个属性用到哪个就设置哪个，不一定全部使用。有的节点代表具体的文件，有的节点只用来描述层次结构。

定义站点地图后，就可以利用 SiteMapPath 控件在页面中实现导航功能。

（4）在解决方案中，分别添加名为“index.aspx”、“teacher.aspx”、“student.aspx”、“course.aspx”的网页。

（5）分别切换到 index.aspx 和 teacher.aspx 的“设计”视图，向页面内拖放一个 SiteMapPath 控件，即可看到该页面在网站内的导航路径。如图 4.27、图 4.28 所示。

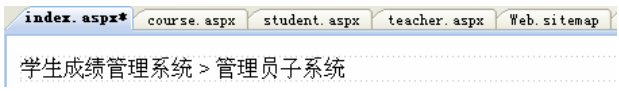


图 4.27　index.aspx 导航效果图

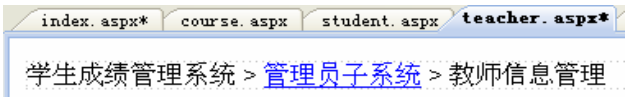


图 4.28　teacher.aspx 导航效果图

（6）在浏览器中查看 teacher.aspx，单击“管理员子系统”超链接，将链接到管理员子系统主页面 index.aspx。

L4.2　美化管理员子系统主界面

L1. 主题

主题是指页面和控件外观属性设置的集合，由 ASP.NET 支持的具有特殊含义的文件夹构成。在主题中，可以包含各种页面外观控制文件和资源文件，主要有：皮肤文件（扩展名为.skin）、级联样式表文件（扩展名为.css）、脚本文件（扩展名为.js）、资源文件（扩展名为.resx）、图像文件、声音文件等。

利用主题可以很方便地统一控制页面外观，把所有与页面外观有关的控制文件和资源文件放在主题文件夹中，页面只需切换主题，主题文件夹下所有的控制文件和资源文件就会自动切换。一个站点内创建多套主题，则可以在网站运行的时候动态地切换网站主题，方便地实现网站外观主题的更换。

有两种类型的主题，一种是应用程序主题，另一种是全局主题。

应用程序主题是指在 Web 应用程序的 App_Themes 文件夹下的一个或多个特殊文件夹，主题的名称是文件夹的名称。本章所提到的主题都是应用程序主题。

如果一个服务器中有多个 Web 应用程序，也可以定义全局主题。全局主题是指保存在服务器特定文件夹下的一个或多个特殊文件夹，具体保存到哪个文件夹由不同的服务器所决定。

1．创建和应用主题的基本步骤

（1）在“解决方案资源管理器”中，右击项目名，选择“添加”→“添加 ASP.NET 文件夹”→“主题”菜单项，系统就会自动判断是否已经存在 App_Themes 文件夹，如果不存在该文件夹，就自动创建，并在该文件夹下添加一个主题文件夹；如果已经存在该文件夹，就直接在该文件夹下添加新的主题文件夹。

（2）在主题文件夹下添加外观控制文件和资源文件。

（3）打开.aspx 文件，切换到“设计”视图，右击选择“属性”选项，在“属性”窗口顶部下拉列表中，选中“Document”文档对象，在列表中定位到“StyleSheetTheme”，设置其值为某个主题名称，本页面就会自动套用主题内的外观控制文件和资源文件。

2．给整个网站应用主题

每个应用程序中都包括多个页面，为了保证和谐统一的用户界面，可以让所有页面使用同一主题。如果在每个页头都设置相同的 StyleSheetTheme 属性值，那么非常麻烦。为了快速地为整个应用程序中所有页面设置相同的主题，可以设置 Web.Config 文件的<pages>配置节内容：

```
<configuration>
    <system.web>
      <page theme="sampleTheme" />
    </system.web>
</configuration>
```

L2. 样式表

样式表（Cascading Style Sheets，CSS）是 W3C 协会为弥补 HTML 在显示属性上的不足而制定的一套样式标准。CSS 标准中重新定义了 HTML 中原来的文字显示样式，并增加了一些新概念（如类、层等），提供了更为丰富多彩的显示样式。CSS 可对网页的样式进行集中管理，它允许将样式定义单独存储于样式文件中，这样可以实现页面的结构与表现分离，便于多个 HTML 文件共享样式定义。

在 XHTML 中，每个标记都可称为一个元素。元素是构成 XHTML 源代码的基本单位，一般用“<元素名称>”开头，用“</元素名称>”结束。例如，<body>…</body>， 等，有些元素也可以以“<元素名称 />”的形式出现，如
。

样式是指元素在浏览器中的呈现形式，如元素的高度、宽度、是否有边框、边框颜色、边框粗细、字体大小、字体颜色、元素的背景色和背景图片、元素内数据的对齐方式等。在 XHTML 中可以通过 style 属性设置元素的样式，每个 style 内包含一个或多个属性，其一般形式为：

```
<元素 style="属性名 1: 属性值 1; 属性名 2: 属性值 2; ……">
```

属性名与属性值之间用冒号“:”分隔，如果一个样式有多个属性，各属性之间用分号“;”隔开。

L3. 样式选择符

CSS 规定了 3 种定义样式的方式：一是直接将样式放置在元素内，通过 style 属性设置，称为内联式；二是在网页的 head 部分定义样式，称为嵌入式；三是以扩展名为.css 的文件保存样式定义，称为外部链接式。三种方式分别适用于不同的场合。

内联式适用于单独控制某个元素样式的情况。这种方式的优点是设置样式直观、方便；缺点是修改某些元素的样式时，需要打开网页文件。

下边的代码采用内联式控制各个元素的样式：

```
<body style="background-color: #8888ff; text-align: center;">
    <div style="text-align: center;" id="div0">
        <img src="njnu.gif" alt="图片找不到" style="width: 300px; height: 60px;"/><br/>
        <div style="font-size: 40px; color: red;" id="div1">学生 成绩管理系统<br /></div>
    </div>
</body>
```

嵌入式适合控制一个网页内具有相同样式的多个元素。采用这种方式的优点是当修改某些元素的样式时，只需要修改 head 部分的样式规则即可，该网页内所有使用了该样式规则的元素都会自动应用新的样式。但这种方式仅仅适用于修改某个网页内的具有相同样式的元素，如果多个网页的很多元素均采用相同的样式，仍然需要在各个网页的 head 部分重复定义相同的样式规则。

如果把上例中的样式改成嵌入式定义，在 head 内定义样式，可以将代码修改如下：

```
<head>
    <title>一个 Hello 网页</title>
    <style type="text/css">
        #div0{text-align: center;}
        img{width: 300px; height: 60px;}
        #div1{font-size: 40px; color: red;}
    </style>
</head>
<body style="background-color: #8888ff; text-align: center;">
    <div id="div0">
        <img src="njnu.gif" alt="图片找不到" id="img"/><br/>
        <div id="div1">学生 成绩管理系统<br /></div>
    </div>
</body>
```

外部链接式适用于控制多个网页内具有相同样式的元素。这种方式将样式保存在一个或多个单独的.css 文件中，当需要修改元素的样式时，只需要修改.css 文件中的样式规则即可。一旦修改.css 文件中的某个样式规则，凡是引用了.css 内修改过的样式规则的元素，都

会自动应用新的样式。

.css 文件内的内容和嵌入式方式下 head 内的 style 标记内的内容相同，只是单独保存在一个文件中，例如：

```
#div0{text-align: center;}
img{width: 300px; height: 60px;}
#div1{font-size: 40px; color: red;}
```

在 XHTML 中引用样式文件后，文件内的元素才会应用样式文件内的样式规则。引用样式文件的方法：在 head 标记块内添加下面的代码。

```
<link href="StyleSheet.css" rel="stylesheet" type="text/css" />
```

其中，rel 属性规定了 XHTML 与被链接文件的关系，type 属性规定了链接文件的类型，href 属性规定了要链接的样式文件的 URL。

注：在某网页“设计”视图下，打开“属性”栏，在“顶部”下拉列表中，选择“DOCUMENT”，在属性列表中找到“StyleSheet”属性，单击后会出现文件浏览窗口，通过文件浏览定位到 css 文件，就可以通过可视化方法给文件添加对 css 文件的引用。设置完后会自动在 head 内生成代码。

因此，比较好的方式是将样式放在单独的 css 文件中，然后在每个网页内添加对它的引用。

页面采用内联式的时候，样式直接定义在元素的 style 属性上，所定义的样式只对某个元素起作用。当页面采用嵌入式和外部链接式时，样式是单独定义的，把这种单独定义的样式称为样式规则。

样式规则必须符合如下格式：

样式选择符{ 样式属性 1:值 1； 样式属性 2:值 2； 样式属性 3:值 3; …… }

其中包含两个部分：样式选择符，样式属性。

样式选择符用来设置样式所作用的元素范围。 { }内是所采用的样式。{ }内的样式会对样式选择符作用范围内的所有元素有效。样式属性可以通过可视化的“样式生成器”生成，它和 sytle 属性的值相同，因此理解样式规则的关键是样式选择符。

为了方便控制样式规则所作用的元素范围，CSS 中把样式选择符分为如下几类：标签选择符、类选择符、ID 选择符、虚类选择符、包含选择符和并列选择符。

1．标签选择符

标签选择符（即 HTML 或 XHTML 标记）是最典型的选择符类型。定义的时候直接使用标记名称作为选择符名称。常见格式如下：

标签名{ 样式属性 1:值 1； 样式属性 2:值 2； 样式属性 3:值 3; …… }

例如：

```
div
{
    background-color: white;     //背景颜色
    text-align: center;          //文本水平布局
}
```

引用该 css 文件的页面内所有 div 标记会采用{}内的样式。

2．类选择符

标签选择符比较方便，能够对某种标签定义样式，但是有的时候并不想对网页内所有此标签应用同一种样式，而只想对某些标签应用某种样式，它们不一定会是相同的标签。常见形式如下：

.类名{ 样式属性 1:值 1; 样式属性 2:值 2; 样式属性 3:值 3; …… }

例如：

```
.center
{
    background-color: white;      //背景颜色
    text-align: center;           //文本水平布局
}
```

如果某个标签的 class 属性设置为类选择符的名称，类选择符就会对此标签起作用。例如：

```
<h1 class="center">类选择符</h1>
<h2 class="center">类选择符</h2>
```

h1 和 h2 的 class 属性都设置为 center，则 center 样式规则就会对它们起作用。

注：类名的前面有个“.”,表示是自定义类。利用样式生成器定义的时候，系统会自动加上这个点，但引用的时候不要带点。

3．ID选择符

ID 选择符用于对某个元素定义样式规则，只能用于某个元素。常见格式如下：

＃ID 名称{ 样式属性 1:值 1; 样式属性 2:值 2; 样式属性 3:值 3; …… }

例如：

```
#header
{
    background-color: white;
    text-align: center;
}
```

如果某个标签的 ID 属性的值和 ID 选择符的名称相同，则 ID 选择符会对此标签生效。例如：

```
<div id="header"> ... </div>
```

可以看出类选择符和 ID 选择符定义方式非常相似，但也存在一定的区别。多个标签可以使用同一个自定义类，而 ID 却只能用于某一个标签。

利用 CSS 定义样式时还要注意，如果某个元素同时受标签选择符、类选择符和 ID 选择符的影响，要注意它们的优先级问题：ID 选择符的优先级最高，其次是类选择符，标签选择符优先级最低。

4．伪类

“伪类”是专门针对超链接标签的选择符，使用伪类可以为访问过的、未访问过的、激

活的、鼠标指针悬停于其上的 4 种状态超链接定义不同的显示样式：

A:link：代表未访问过的超链接。

A:visited：代表访问过的超链接。

A:active：当超链接处于选中状态。

A:hover：当鼠标指针移动到超链接上。

例如：

```
A:active
{
    color: blue;
    background-color: buttonface;
}
```

5．包含选择符

包含选择符用于定义具有层次关系的样式规则，它由多个样式选择符组成，一般格式如下：

选择符 1　选择符 2…{ 样式属性 1:值 1; 样式属性 2:值 2; 样式属性 3:值 3; …… }

各选择符之间用空格分割，例如：

```
DIV  P  H1
{
    font-weight: bold;
    color: red;
}
```

表示只有处于 DIV 标签内的 P 标签内的 H1 标签才会应用如上的样式，其他地方的 H1 不会受此样式影响。

L4. 样式生成器

每个元素可以设置的样式有很多，对于初学者不可能很快就掌握，因此 Visual Studio 2008 集成了一个图形化的“样式生成器”窗口，可以帮助开发人员快速地对元素的样式进行设置。对于初学者，只需掌握样式生成器中每种设置所对应的效果即可，无须完全掌握 CSS 中所有样式。

样式生成器可以帮助设计者快速地设置元素的样式，在“设计”视图下，选中某个元素，右击选择“属性”，在“属性”窗口中选择“style”，即可出现如图 4.29 所示“修改样式”窗口。

注：Visual Studio 2008 为了便于元素的选择，提供了 HTML 标签导航，如图 4.30 所示，用于显示目前鼠标所在区域所使用的 HTML 标签，在 HTML 标签导航单击标签，就可以选中该元素。如果在“设计”视图中不方便精确选择，可以打开“属性”窗口，再切换到“源”视图，把光标定位到某个元素内。“属性”窗口中就会显示对应元素的属性。

该对话框分为两个窗格。左窗格列出 9 种常规类别（字体、块、背景、边框、方框、定位、布局、列表、表格），基本包括了 CSS 样式中常用的类别。当选择某种类别时，右窗格会显示所选类别下的相关样式，在窗口的下部有一个“预览”区域，用来显示元素设置样

式后的效果，最下部有个“说明”区域，用来显示所设置的 CSS 代码。设置完样式选项后，单击“确定”按钮，样式生成器会自动给对应的元素生成对应的样式定义。

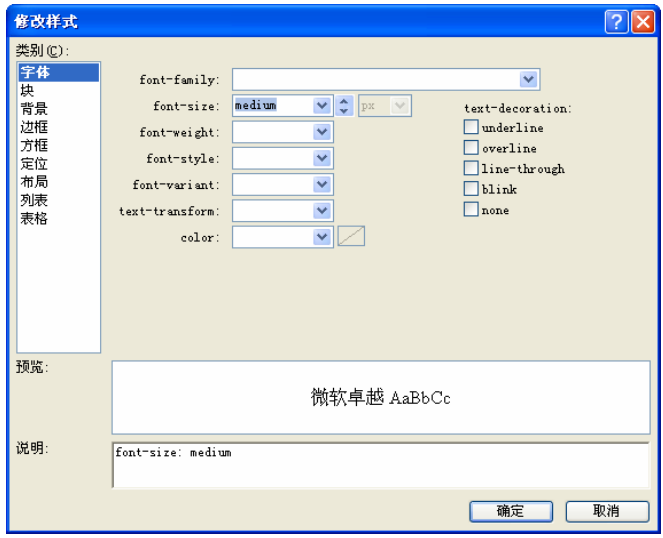


图 4.29　样式设计器－字体

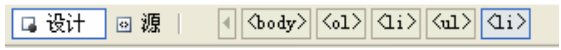


图 4.30　HTML 标签导航

下面就常规类别中的样式功能进行介绍。

1．字体

如图 4.29 所示，字体类别主要用来对元素内的文本的字体进行相应的设置，可以设置字体名称（font-family），字体大小（font-size），字体粗细（font-weight），字体风格（font-style），小型大写字母字体（font-variant），字母转换（font-transform），字体颜色（color），字体装饰（text-decoration）。

（1）font-family：设置元素内字体所采用的字体名称，如“宋体”、“黑体”等。

（2）font-size：设置元素内字体的大小，字体的大小采用 CSS 中统一的长度单位。

字体大小的取值很多，可能取值如表 4.3、表 4.4 所示。

表 4.3　font-size 取值

值	描　述
xx-small、x-small、small、medium large、x-large、xx-large	把字体的尺寸设置为不同的尺寸，从 xx-small 到 xx-large。默认值为 medium。medium 和浏览器的基准字号相同（基准字号默认是 16 px）
smaller	把 font-size 设置为比父元素更小的尺寸
larger	把 font-size 设置为比父元素更大的尺寸
inherit	把 font-size 设置为父元素的尺寸
%	把 font-size 设置为基于父元素的一个百分比
值	把 font-size 设置为某个确定的值

表 4.4　CSS 长度单位

值	描　述	值	描　述	值	描　述
px	像素	pt	点（1 pt=1/72 in）	pc	派卡（1 pc=12 pt）
cm	厘米	mm	毫米	in	英寸
ex	当前浏览器默认字体中小写字母“x”的高度	em	当前浏览器字体中大写字母“M”的宽度		

（3）font-weight：设置元素内字体的粗细，可能取值如表 4.5 所示。

表 4.5　font-weight 取值

值	描　述
normal	默认。定义标准的字符
bold	定义粗体字符
bolder	定义更粗的字符
lighter	定义更细的字符
100 200 300 400 500 600 700 800 900	定义由细到粗的字符。400 等同于 normal，而 700 等同于 bold

（4）font-style：设置元素内字体的风格，可能的取值如表 4.6 所示。

表 4.6　font-style 取值

值	描　述
normal	默认。浏览器显示标准的字体
italic	浏览器显示斜体的字体
oblique	浏览器显示倾斜的字体

（5）font-variant：设置元素内小型大写字母的显示方式，可能的取值如表 4.7 所示。

表 4.7　font-variant 取值

值	描　述
normal	默认。浏览器显示一个标准的字体
small-caps	浏览器显示小型大写字母的字体，这意味着所有的小写字母均会被转换为大写，但是所有使用小型大写字体的字母与其他文本相比，其字体尺寸更小

（6）font-transform：设置元素内单词的字母转换，可能的取值如表 4.8 所示。

表 4.8　font-transform 取值

值	描　述
none	默认，无转换发生
capitalize	将每个单词的第一个字母转换成大写，其他无转换发生
uppercase	转换成大写
lowercase	转换成小写

（7）color：设置元素内字体的颜色。通过可视化的颜色选择器进行选择即可。

（8）text-decoration：设置字体的装饰，可能的取值如表 4.9 所示。

表 4.9　text-decoration 取值

值	描　述	值	描　述
none	默认，无装饰	line-through	贯穿线
blink	闪烁	overline	上划线
underline	下划线		

2．块

如图 4.31 所示，块主要用来对文本块进行设置，可以设置块内文字的行高（line-height），文字垂直对齐方式（vertical-align），文字水平对齐方式（text-align），文字缩进（text-indent），空格字符处理方式（white-space），单词间距（word-spacing），文字间距（letter-spacing）。

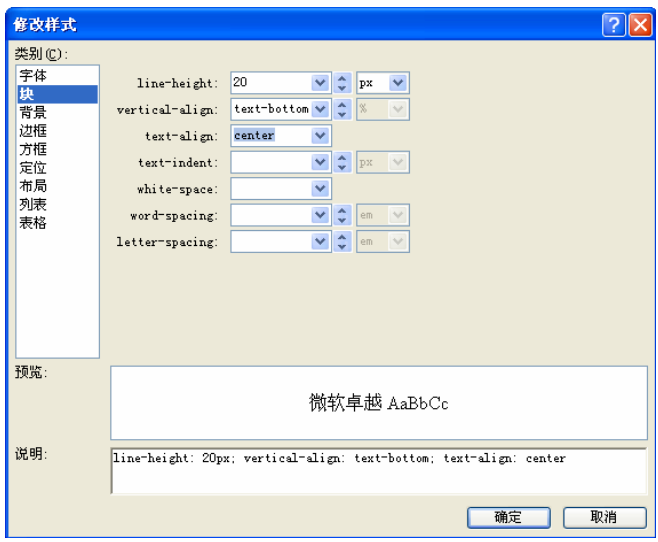


图 4.31　样式设计器－块

（1）line-height：设置元素内文字的行高，字体最底端与字体内部顶端之间的距离，取值为“由浮点数字和单位标识符组成的长度值 | 百分数”。

（2）vertical-align：设置元素内文字的垂直对齐方式，可能的取值如表 4.10 所示。

表 4.10　vertical-align 取值

值	描　述
sub	垂直对齐文本的下标
super	垂直对齐文本的上标
值	由浮点数字和单位标识符组成的长度值 | 百分数。可为负数。定义由基线算起的偏移量。基线对于数值来说为 0，对于百分数来说就是 0%

（3）text-align：设置元素内文字的水平对齐方式，可能的取值如表 4.11 所示。

表 4.11　text-align 取值

值	描　述
left	默认值。左对齐
right	右对齐
center	居中对齐
justify	两端对齐

（4）text-indent：设置元素内文字的缩进，取值为“由浮点数字和单位标识符组成的长度值 | 百分数”。

（5）white-space：设置元素内空格字符处理方式，可能的取值如表 4.12 所示。

表 4.12　white-space 取值

值	描　述
normal	默认处理方式。文本自动处理换行。假如抵达容器边界，内容会转到下一行
pre	用等宽字体显示预先格式化的文本。不合并字间的空白距离和进行两端对齐
nowrap	强制在同一行内显示所有文本，直到文本结束或遇到 br 元素

（6）word-spacing：设置元素内单词间的间距，取值为“由浮点数字和单位标识符组成的长度值 | 百分数”。

（7）letter-spacing：设置元素内文字间的间距，取值为“由浮点数字和单位标识符组成的长度值 | 百分数”。

3．背景

如图 4.32 所示，背景类别主要用来对元素的背景进行相应的设置，可以设置背景颜色（background-color），背景图片（background-image），背景图片的平铺方式（background-repeat），背景图片的滚动方式（background-attachment），背景图片在元素内的 X 轴与 Y 轴的相对位置（background-position）等。

（1）background-color：设置元素背景颜色，通过颜色选择窗口选择。

（2）background-image：设置元素背景图片，通过“浏览”定位到某张图片。

（3）background-repeat：设置元素背景图片的平铺方式，常用于背景图片大小小于元素的大小的情况，可能取值如表 4.13 所示。

表 4.13　background-repeat 取值

值	描　述
repeat	默认值。背景图像在纵向和横向上平铺
no-repeat	背景图像不平铺两端对齐
repeat-x	背景图像仅在横向上平铺
repeat-y	背景图像仅在纵向上平铺

（4）background-attachment：设置元素背景图片是否随元素内容滚动，对于那些带滚动条的元素需要设置，可能取值如表 4.14 所示。

表 4.14　background-attachment 取值

值	描　　述
scroll	默认值。背景图像随元素内容的滚动而滚动
fixed	背景图像固定

（5）background-position：设置背景图片在元素内的位置。分为 X 轴上的位置和 Y 轴上的位置，元素的左上角为坐标原点，顶部边缘往右的方向为 X 轴方向，左部边缘向下的方向为 Y 轴方向。它们可能的取值如表 4.15 所示。

表 4.15　(x),(y)background-position 取值

值	描　　述
left	居左
center	居中
right	居右
值	百分数 | 由浮点数字和单位标识符组成的长度值

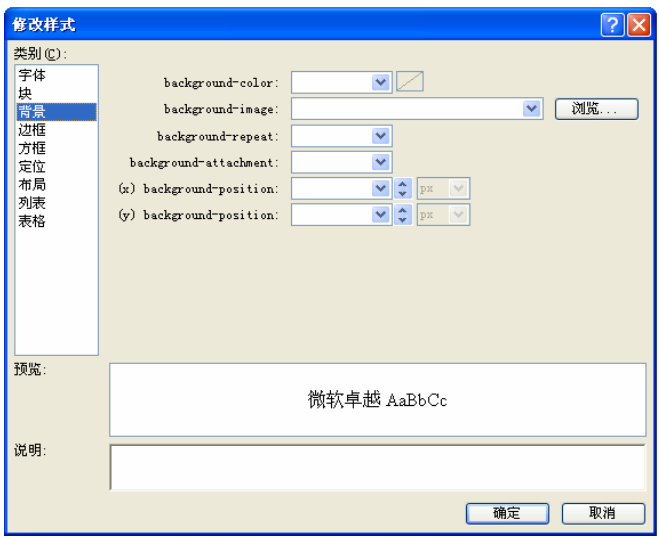


图 4.32　样式设计器－背景

4．边框

如图 4.33 所示，用来设置元素四周的边框样式（border-style），边框宽度（border-width），边框颜色（border-color）。

如果设置的时候选中“全部相同”，则 4 个边框都采用相同的设置；如果取消选中“全部相同”，则可以对不同的边框设置不同的值。

（1）border-style：设置元素边框的样式，可能取值如表 4.16 所示。

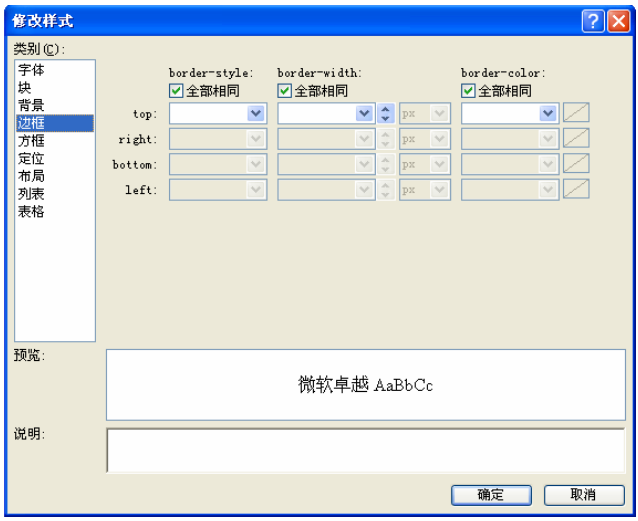


图 4.33　样式生成器－边框

表 4.16　border-style 取值

值	描　　述
none	默认值。无边框。不受任何指定的 border-width 值影响
hidden	隐藏边框。IE 不支持
dotted	在 MAC 平台上 IE4+与 Windows 和 UNIX 平台上 IE5.5+为点线。否则为实线
dashed	在 MAC 平台上 IE4+与 Windows 和 UNIX 平台上 IE5.5+为虚线。否则为实线
solid	实线边框
double	双线边框。两条单线与其间隔的和等于指定的 border-width 值
groove	根据 border-color 的值画 3D 凹槽
ridge	根据 border-color 的值画 3D 凸槽
inset	根据 border-color 的值画 3D 凹边
outset	根据 border-color 的值画 3D凸边

（2）border-width：设置元素边框的宽度，可能取值如表 4.17 所示。

表 4.17　border-width 取值

值	描　　述
medium	默认值。默认宽度
thin	小于默认宽度
thick	大于默认宽度
值	由浮点数字和单位标识符组成的长度值，不可为负值

（3）border-color：设置元素边框的颜色，通过颜色选择窗口选择。

5．方框

如图 4.34 所示，方框类别用来设置元素边框与其他元素边框之间距离，称为边距（Margin）。

设置元素内容与元素边框之间距离，称为空白（Padding）。

边距（Margin）可以为正值也可以为负值，负值时，两个元素就可以重叠在一起。空白只能取正值。

如果设置的时候选中“全部相同”，则 4 个边框都采用相同的设置；如果取消选中“全部相同”，则可以对不同的边框设置不同的值。

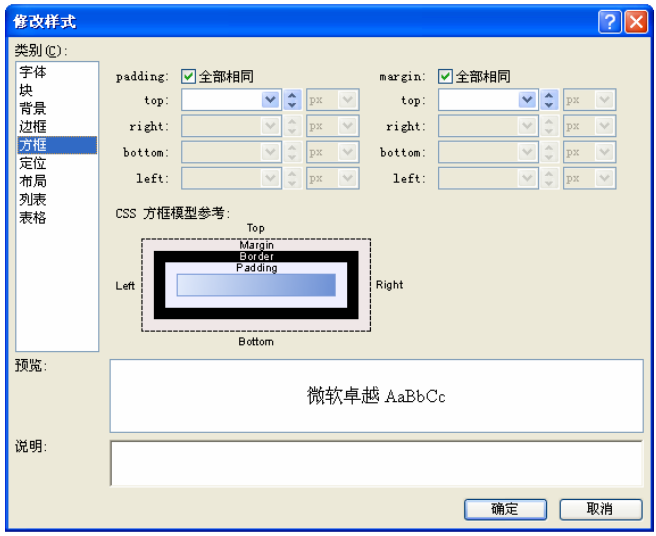


图 4.34　样式控制器－方框

要理解元素边距和空白的概念，需要理解 CSS 中元素的“盒模型”。网页内的每一个元素在网页上都是可以看成一个盒子，盒子结构如图 4.35 所示。即一个元素由如下几部分组成的：内容、空白、边框、边距。因此一个元素的实际长度等于：左边距＋左边框＋左空白＋内容长度＋右空白＋右边框＋右边距。通过元素的 width 和 height 属性所设置的元素高度和宽度是元素内容的高度和宽度，因此元素在页面内所占用的高度和宽度要比设置的大。

若要让元素与边框之间有空隙，可以通过元素的“空白”设置；想让两个元素之间有一定的距离，可以通过元素的“边距”设置。

图 4.35　CSS 盒子模型

6．定位

如图 4.36 所示，定位类别主要用来设置元素在页面中的位置属性。包括位置模式（position），层次顺序（z-index），宽度（width），高度（height），相对具有定位设置的父元素的位置关系（top、right、bottom、left）。

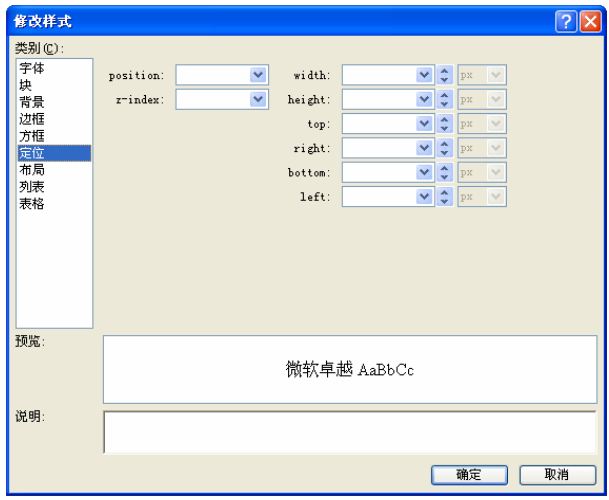


图 4.36　样式控制器－定位

position：用来设置元素在网页内的位置模式，可能取值如表 4.18 所示。

表 4.18　position 取值

值	描　　述
static	默认值。无特殊定位，元素遵循 HTML 定位规则，正常流中的位置
absolute	将元素从文档流中拖出，使用 top、right、bottom，left 等属性相对于其最接近的一个最有定位设置的父元素进行绝对定位。如果不存在这样的父元素，则依据 body 元素。允许元素重叠，层叠通过 z-index 属性定义
relative	元素不可层叠，但将依据 top、right、bottom、left 等属性在正常文档流中偏移位置
fixed	未支持。元素定位遵从绝对（absolute）方式，但是要遵守一些规范

不同位置模式的定位原则如下。

（1）正常流中的位置

正常流中的位置是按照如下的布局显示的。网页中的元素按照从左到右、从上到下的顺序显示，各元素之间不重叠。如果不设置元素定位方式，则正常流中的位置作为默认显示方式。

（2）绝对定位

绝对位置是指元素显示在页中的位置由 sytle 样式的 top、right、bottom、left 及 z-index 属性决定的。

z-index 值大的元素会覆盖 z-index 值小的元素，即 z-index 值大的元素会显示在上层。

为了控制“绝对定位”元素在网页上的位置不随客户端分辨率改变而改变，通常用定位模式为“正常流中的偏移量”的元素作为“绝对定位”的元素的参考物。

（3）正常流中的偏移量

正常流中的偏移量是元素在正常流中的位置基础上，再做一定的偏移后所处的位置为元素的显示位置。如果元素在正常流中的位置做了偏移，那以前应该在的位置会出现一个空洞，别的元素无法占用。因此一般只把一个元素的模式设置为“正常流中偏移量”，而实际

不进行位置偏移。

top 是元素上边框离参考物上边框的距离，right 是元素右边框离参考物右边框的距离，bottom 是元素下边框离参考物下边框的距离，left 是元素左边框离参考物左边框的距离。

top、right、bottom、left 的取值为“由浮点数字和单位标识符组成的长度值 | 百分数”，可以取负值。

top、right、bottom、left 属性的值只有在 absolute 和 relative 位置模式下才有效。

width 和 height 是元素的宽度和高度。当设置了 width 和 height 属性的值，又设置 top、right、bottom、left 属性的值的情况下，则元素的位置只受 width、height、top 和 left 属性的约束。

7．布局

如图 4.37 所示，布局类别主要用来设置元素在网页内的布局属性。包括可见性（visibility），显示方式（display），浮动（float），浮动清除（clear），光标形状（cursor），溢出控制（overflow），可视区域（clip：rect（…））。

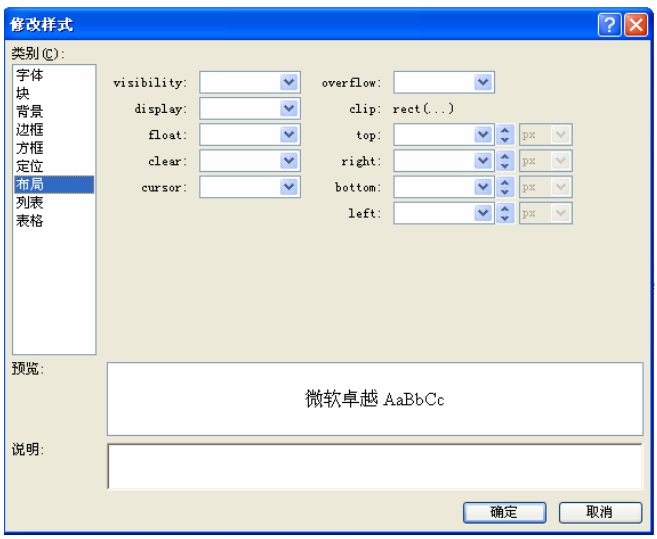


图 4.37　样式控制器－布局

（1）visibility：设置元素的可视属性，可能取值如表 4.19 所示。

表 4.19　visibility 取值

值	描　　述
visible	元素可视
hidden	元素隐藏
collapse	主要用来隐藏表格的行或列。隐藏的行或列能够被其他内容使用

（2）display：设置元素是否显示及如何显示，可能取值如表 4.20 所示。

表 4.20　display 取值

值	描　　述
block	默认值。将元素强制作为块元素呈递，为元素之后添加新行
none	隐藏元素。与 visibility 属性的 hidden 值不同，其不为被隐藏的元素保留物理空间
inline	内联元素的默认值。将元素强制作为内联元素呈递

（3）float：设置元素相对于父元素的浮动，可能取值如表 4.21 所示。

表 4.21　float 取值

值	描　　述
left	元素浮动到父元素的左边
right	元素浮动到父元素的右边

（4）clear：设置元素两边是否允许有浮动元素，可能取值如表 4.22 所示。

表 4.22　clear 取值

值	描　　述
both	父元素内，元素的两面都不允许有元素
left	父元素内，元素的左边不允许有元素
right	父元素内，元素的右边不允许有元素

（5）cursor：设置光标停留到元素内的形状。

（6）overflow：设置元素内容溢出情况下的显示方式，可能取值如表 4.23 所示。

表 4.23　overflow 取值

值	描　　述
visible	默认值。不剪切内容，也不添加滚动条
auto	在必须时对象内容才会被裁切或显示滚动条
scroll	总是显示滚动条
hidden	不显示超过对象尺寸的内容

（7）clip：设置元素的可视区域，通过一个矩形区域来指定。

8．列表

如图 4.38 所示，列表类别主要用来设置元素内的列表元素的列表样式。包括列表标记样式（list-style-type），列表标记图片（list-style-image），列表标记位置（list-style-position）。只有元素内有列表元素的时候，列表元素的列表样式才受影响。可以通过“列表”设置有序列表和无序列表的样式。

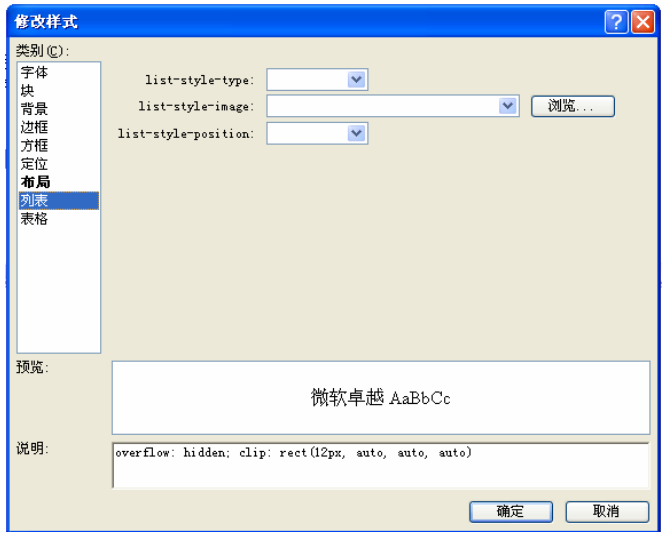


图 4.38　样式控制器－列表

（1）list-style-type：设置元素内列表项所使用的预设标记，可能取值如表 4.24 所示。

表 4.24　list-style-type 取值

值	描　述
disc	默认值。实心圆
circle	空心圆
square	实心方块
decimal	阿拉伯数字
decimal-leading-zero	以一个 0 开头的数字，如 01, 02, 03…
lower-roman	小写罗马文字 i, ii, iii, iv, v, etc.
upper-roman	大写罗马文字 I, II, III, IV, V, etc.
lower-latin	小写拉丁文 a, b, c, …, z
upper-latin	大写拉丁文 A, B, C, …, Z

（2）list-style-image：设置元素内列表项所使用的图片。

第 5 章　学生信息注册与系统登录

◇ 任务目标

本章主要目标是完成学生成绩管理系统的学生信息注册与系统登录模块，学生信息注册模块提供了学生注册功能，方便学生注册。系统登录模块能够实现为不同的角色提供统一的登录界面。当用户登录系统后，就可以访问需要权限认证才可进入的页面，无须重新登录，提供了访问系统的方便性，确保了系统的安全性。

设计好的学生信息注册模块如图 5.1、图 5.2 所示。

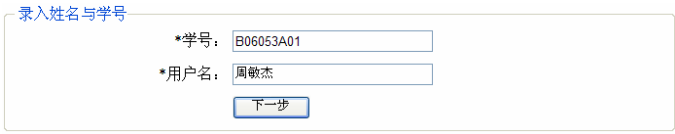


图 5.1　用户注册页面（1）

图 5.2　用户注册页面（2）

设计好的登录页面效果如图 5.3 所示。

图 5.3　系统登录页面效果

第一部分　应 用 实 践

5.1　学生信息注册

1．学生信息注册页面的布局

为了方便学生的注册，把学生的注册页面分为3 个视图完成，第 1 个视图学生录入自己的学号和姓名，第 2 个视图学生录入自己的个人信息和密码保护信息，第 3 个视图显示用户录入的信息，并提示用户注册成功。因此用户注册页面的布局应按照如图 5.4 所示设计。

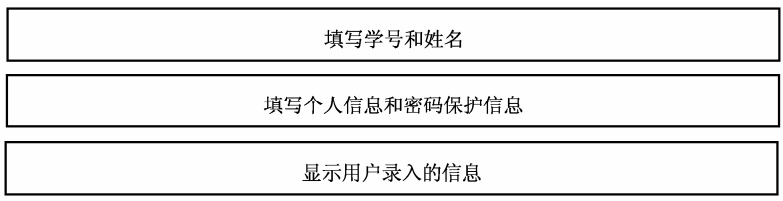


图 5.4　学生信息注册页面的布局

2．设计学生信息注册页面

设计步骤如下所示：

（1）打开创建的“学生成绩管理系统”网站项目，右击“解决方案资源管理器”内的 Students 文件夹，选择“添加新项”选项，在弹出的“添加新项”对话框中，选择“Web 窗体”模板，命名为“StudentRegister.aspx”，选择“Visual C#”语言，不选中“选择母版页”复选框，单击“添加”按钮完成学生信息注册页面的创建。

（2）切换到 StudentRegister.aspx 页面的设计视图，打开属性窗口并选择顶部下拉框中的“Document”文档对象，设置 StyleSheetTheme 属性为“主题 1”，把光标定位到“设计视图”中的 Div 内，打开“应用样式”窗口（选择“视图”→“应用样式”菜单项），单击“应用样式”窗口中的“#innerLayout”样式规则，给该层应用样式规则，打开页面的“源”视图，选中<body>标记，打开“应用样式”窗口，单击“新建样式”按钮，出现“新建样式”窗口，设置“选择器”为“级联样式”，选中“块”类别，设置 text-align 属性为“center”。

（3）切换到 StudentRegister.aspx 页面的源视图，光标定位在 Div 内，双击“工具箱”的“标准”选项卡中的 **MultiView** 控件，并在 MultiView 控件内分别添加 3 个 **View** 控件，添加后视图中的代码如下：

```
<asp:MultiView ID="MultiView1" runat="server">
    <asp:View ID="View1" runat="server">
    </asp:View>
    <asp:View ID="View2" runat="server">
    </asp:View>
    <asp:View ID="View3" runat="server">
    </asp:View>
</asp:MultiView>
```

（4）切换到 StudentRegister.aspx 页面的设计视图，在 3 个 View 中分别添加 **Panel** 控件，3 个 Panel 控件的 GroupingText 属性分别设置为“录入姓名与学号”、“录入基本信息与密码保护信息”和“你的填写信息”。

（5）光标定位到“录入姓名与学号”的 Panel 中，选择“表”→“插入表”菜单项，弹出“插入表格”对话框，设置行数为 2，列数为 3，单击“确定”按钮，添加一个 2 行 3 列的表格。按照同样的方法在“录入基本信息与密码保护信息”Panel 内插入一个 2 行 3 列的表格，在“你的填写信息”Panel 内插入一个 2 行 2 列的表格。

注：当表格行数不够的情况下，可以通过选中一行并右击，选择“插入”→“下面的行”菜单项，则在其下面插入一个新行，因此刚开始的时候只建立 2 行就可以了。

（6）在“解决方案资源管理器”内，打开主题文件夹中的 StyleSheet.css 文件，在其内部加入如下的代码：

```
.ThreeTableOne
{
    width: 33%;
    text-align: right;
    height: 30px;
}
.ThreeTableTwo
{
    width: 33%;
    text-align: left;
    height: 30px;
}
.ThreeTableThree
{
    width: 33%;
    text-align: left;
    height: 30px;
}
.TwoTableOne
```

```
{
        width: 50%;
        text-align: right;
        height: 30px;
}
.TwoTableTwo
{
        width: 50%;
        text-align: left;
        height: 30px;
}
.innerLayout
{
        width: 690px;
        padding: 5px;
        text-align: center;
}
```

（7）选择“录入姓名与学号”Panel 内的 Table 的第一列，通过“应用样式”窗口，给其应用“ThreeTableOne”样式，同样的方法给第二列应用“ThreeTableTwo”样式，第三列应用“ThreeTableThree”样式。“录入基本信息与密码保护信息”Panel 内的 Table 的第一列应用“ThreeTableOne”样式，第二列应用“ThreeTableTwo”样式，第三列应用“ThreeTableThree”样式。“你的填写信息”Panel 内的 Table 第一列应用“TwoTableOne”，第二列应用“TwoTableTwo”。

（8）在“录入姓名与学号”Panel 内的 Table 中按照如图 5.5 所示添加控件，并按照表 5.1 设置相关控件的属性。

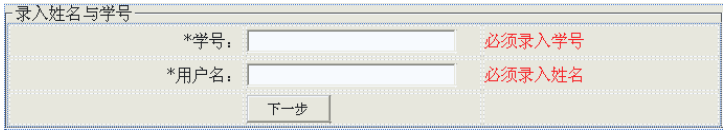


图 5.5　View1 设计

表 5.1　控件属性设置

控 件 类 型	相关属性和值
Textbox	ID=“TextBoxstuID”, Width=“200 px”
Textbox	ID=“TextBoxstuName”, Width=“200 px”
RequiredFieldValidator	ID=“RequiredFieldValidator1” ControlToValidate =“TextBoxstuName” ErrorMessage=“必须录入姓名”
RequiredFieldValidator	ID=“RequiredFieldValidator2” ControlToValidate =“TextBoxstuID” ErrorMessage=“必须录入学号”
Button	ID=“Button1”, Text=“下一步”, Width=“80 px”

（9）在“录入基本信息与密码保护信息”Panel 内的 Table 中按照如图 5.6 所示添加所需控件，相应控件的添加与设置方法见表 5.2。

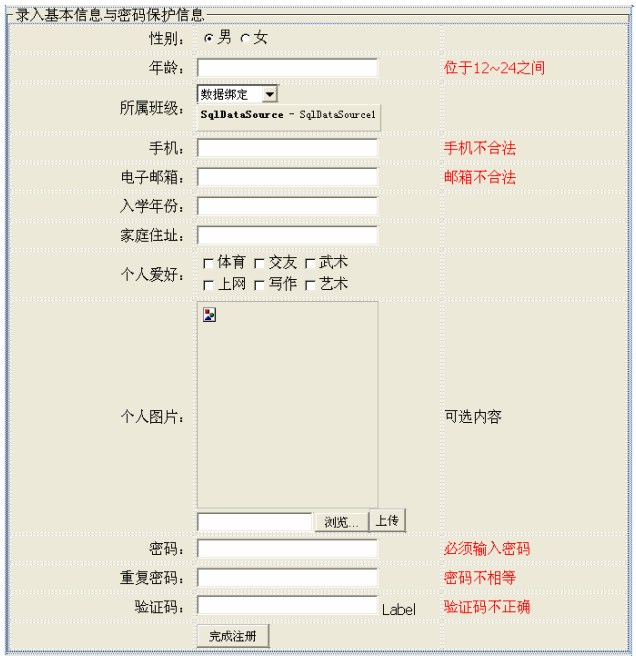


图 5.6　View2 设计

表 5.2　控件属性设置

控 件 类 型	相关属性和值
RadioButtonList	设置 ID 为“RadioButtonListstuSex”，通过 Items 属性，打开“集合编辑器”，添加 2 项：设置第一项的 Text 为“男”，Value 为“F”，Seleted 为“True”；设置第二项的 Text 为“女”，Value 为“M”；RepeatDirection 为“Horizontal”
Textbox	设置 ID 为“TextBoxstuAge”，Width 为“200 px”
RangeValidator	设置 ID 为“RequiredFieldValidator1”，ControlToValidate 为“TextBoxstuAge”，MaximumValue 为“24”，MinimumValue 为“12”，Type 为“Integer”，ErrorMessage 为“位于 12～24 之间”
SqlDataSource	设置 ID 为“SqlDataSource1”，ConnectionString 为“SqlServer2005ConnectionString”，SelectQuery 为“SELECT c.classID, c.className FROM Class c”
DropDownList	设置 ID 为“DropDownListstuClassID”，DataSourceID 为“SqlDataSource1”，DataTextField 为“className”，DataValueField 为“classID”
Textbox	设置 ID 为“TextBoxstuTelphone”，Width 为“200 px”
RegularExpressionValidator	设置 ID 为“RegularExpressionValidator1”，ControlToValidate 为“TextBoxstuTelphone”，ValidationExpression 为“^(13|15|18)\d{9}$”，ErrorMessage 为“手机不合法”
Textbox	设置 ID 为“TextBoxstuEmail”，Width 为“200 px”
RegularExpressionValidator	设置 ID 为“RegularExpressionValidator2”，ControlToValidate 为“TextBoxstuEmail”，ValidationExpression 为“\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*”，ErrorMessage 为“邮箱不合法”

续表

控 件 类 型	相关属性和值
Textbox	设置 ID 为“TextBoxstuEnrollmentTime”，Width 为“200 px”
Textbox	设置 ID 为“TextBoxstuHomeAddress”，Width 为“200 px”
CheckBoxList	设置 ID 为“CheckBoxListstuLove”，添加 6 项，分别设置 6 项的 Text 和 Value 为“体育”、“交友”、“武术”、“上网”、“写作”、“艺术”，RepeatDirection 为“Horizontal”，RepeatColumns 为“3”
Image	设置 ID 为“ImagestuPic”，Width 为“200 px”
FileUpload	设置 ID 为“FileUploadstuPic”
Button	设置 ID 为“ButtonUpload”，Text 为“上传”，CausesValidation 为“False”
Textbox	设置 ID 为“TextBoxstuPassword1”，Width 为“200 px”，TextMode 为“Password”
RequiredFieldValidator	设置 ID 为“RequiredFieldValidator3”，ControlToValidate =” TextBoxstuPassword1”，ErrorMessage=“必须录入密码”
Textbox	设置 ID 为“TextBoxstuPassword2”，Width 为“200 px”，TextMode 为“Password”
CompareValidator	设置 ID 为“CompareValidator1”，ControlToCompare 为“TextBoxstuPassword2”，ControlToValidate 为“TextBoxstuPassword1”，ErrorMessage=“密码不相等”
Textbox	设置 ID 为“TextBoxCode”，Width 为“200 px”
Label	设置 ID 为“LabelCode”
CompareValidator	设置 ID 为“CompareValidator2”，ControlToValidate 为“TextBoxCode”，ErrorMessage 为“验证码不正确”
Button	设置 ID 为“ButtonComplete”，Text 为“完成注册”，Width 为“80 px”

3．编写学生信息注册页面后台代码

添加代码步骤如下：

（1）页面第一次加载的时候，设置 MultiView 默认显示第一个 View，因此需要在 Page 的 Load 事件内添加代码，按【F7】键切换到 StudentRegister.aspx.cs 代码页，添加如下代码：

```
//页面第一次请求的时候执行
if (!Page.IsPostBack)
    this.MultiView1.ActiveViewIndex = 0;  //显示第一个视图
```

（2）当用户填写完学号和姓名后，单击“下一步”按钮，需要判断系统里是否已有重复的学号，如果有则提示用户，没有则设置 MultiView 显示第二个 View，并设置验证码为当前时间，因此需要添加“下一步”按钮的 Click 事件代码，按【F7】键切换到设计视图，双击“下一步”按钮，系统自动切换到代码页并添加了“Button1_Click”方法，在此方法中添加如下代码：

```
    //获取用户录入的信息
    string stuName = this.TextBoxstuName.Text;
    string stuID = this.TextBoxstuID.Text;
    //利用用户录入的信息在表内进行查询
    string sql = "SELECT s.stuID, s.stuName FROM Students
      WHERE s.stuID=@stuID   AND s.stuName=@stuName";
```

```
List<SqlParameter> para = new List<SqlParameter>();
para.Add(commDBHelper.CreateParameters("@stuID",DbType.String,stuID));
para.Add(commDBHelper.CreateParameters("@stuName",DbType.String,stuName));
Boolean isExist = false;
commDBHelper.ExecuteReader(sql, CommandType.Text, out isExist, para);
if (isExist)
{
    //存在，则显示已被注册信息
    Response.Write("<script>alert('已被注册，请联系管理员')</script>");
}
else
{
    //不存在，继续注册，则显示第二个视图
    this.MultiView1.ActiveViewIndex = 1;
    //设置验证控件的显示与需要比较的值
    this.LabelCode.Text = DateTime.Now.Second.ToString();
    this.CompareValidator2.ValueToCompare = this.LabelCode.Text;
}
```

在顶部引入如下的命名空间：

```
using System.Data.SqlClient;
using System.Collections.Generic;
```

（3）当通过文件上传控件浏览选中一个图片后，单击“上传”按钮把需要上传的图片上传到服务器上，并通过 Image 控件显示出所上传的图片。首先在解决方案资源管理器内添加一个命名为“upload”的文件夹用于存放图片，按【F7】键切换到设计视图，双击“上传”按钮，添加“上传”按钮的 Click 事件代码，代码如下所示：

```
/*利用 FileUpload 控件的 PostedFile 属性的 SaveAs 方法进行保存，
保存的路径应该是服务器端的绝对路径*/
//利用 Server.MapPath 实现相对路径与绝对路径的转换
this.FileUploadstuPic.PostedFile.SaveAs(Server.MapPath("~/upload/") +
this.FileUploadstuPic.FileName);
//把上传的文件的相对地址赋值给 image 控件
this.ImagestuPic.ImageUrl = "~/upload/" + this.FileUploadstuPic.FileName;
```

（4）填写完所有信息后，单击“注册完成”按钮，实现把学生所填写的所有信息添加到数据库中。按【F7】键切换到设计视图，双击“注册完成”按钮，添加 Click 事件代码，代码如下所示：

```
//创建参数并赋值
List<SqlParameter> para = new List<SqlParameter>();
para.Add(commDBHelper.CreateParameters("@stuID",DbType.String,this.TextBoxstuID.Text));
para.Add(commDBHelper.CreateParameters("@stuName", DbType.String, this.TextBoxstuName.Text));
```

```
para.Add(commDBHelper.CreateParameters("@stuSex", DbType.String,
                                        this.RadioButtonListstuSex.SelectedValue));
para.Add(commDBHelper.CreateParameters("@stuAge", DbType.String, this.TextBoxstuAge.Text));
para.Add(commDBHelper.CreateParameters("@stuClassID", DbType.String,
                                        this.DropDownListstuClassID.SelectedValue));
para.Add(commDBHelper.CreateParameters("@stuEmail", DbType.String,this.TextBoxstuEmail.Text));
para.Add(commDBHelper.CreateParameters("@stuTelphone", DbType.String,
                                        this.TextBoxstuTelphone.Text));
para.Add(commDBHelper.CreateParameters("@stuEnrollmentTime", DbType.String,
                                        this.TextBoxstuEnrollmentTime.Text));
para.Add(commDBHelper.CreateParameters("@stuHomeAddress", DbType.String,
                                        this.TextBoxstuHomeAddress.Text));
para.Add(commDBHelper.CreateParameters("@stuPassword", DbType.String,
                                        commTools.toMD5(this.TextBoxstuPassword1.Text)));
//构建出学生的个人爱好
string love = "";
for (int i = 0; i < this.CheckBoxListstuLove.Items.Count; i++)
{
    if (this.CheckBoxListstuLove.Items[i].Selected)
    {
        love += this.CheckBoxListstuLove.Items[i].Text + ";";
    }
}
love = love.Trim(';');                                  //删除最后一个“;”字符
para.Add(commDBHelper.CreateParameters("@stuLove", DbType.String, love));
para.Add(commDBHelper.CreateParameters("@stuPic", DbType.String, this.ImagestuPic.ImageUrl));
//执行插入操作
commDBHelper.ExecuteNonQuery("insertStudent", CommandType.StoredProcedure, para);
```

4．在浏览器中使用本页

（1）在浏览器中查看本页，首先显示如图 5.7 所示的界面。

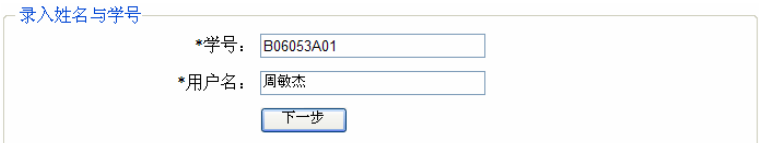


图 5.7　学生注册界面（a）

（2）分别录入用户名和学号，单击“下一步”按钮，则出现如图 5.8 所示的录入详细信息界面：

图 5.8　学生注册界面（b）

（3）单击“完成注册”按钮，把录入的信息添加到数据库内。

5.2　系统登录

1．设计系统登录布局

登录界面的布局设计如图 5.9 所示。

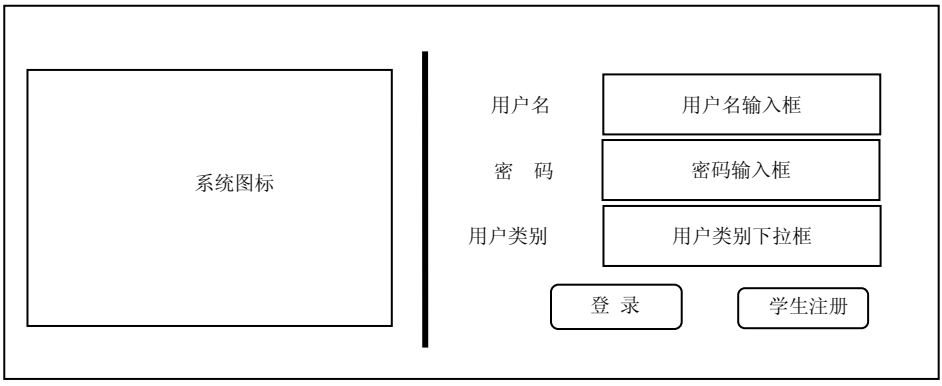


图 5.9　登录界面的布局设计

2．设计系统登录页面

本系统用站点默认建立的 Default.aspx 页面作为系统的登录界面，初次访问本站点，首先会显示本页面。因此在 ISS 中需要把默认首页配置为“Default.aspx”。

设计步骤如下：

（1）打开前面所创建的网站项目，在“资源管理器解决方案”中右击“Default.aspx”，选择“视图设计器”，选中页面中的 div，通过“属性”窗口设置其 id 为“main”。

（2）选择“视图”→“应用样式”菜单项，打开“应用样式”窗口。单击窗口中的“新建样式”选项，在弹出的“新建样式”对话框中的选择器框中输入“#main”，选择“定位”类别，设置 position 为“absolute”，width 为“700 px”，height 为“180 px”，top 为“50%”，left 为“50%”。选择“方框”类别，设置 margin 的 top 为“–90 px”，right 和 bottom 为“0 px”，left 为“–350 px”。单击“确定”按钮，会在 Default.aspx“源”视图顶部生成如下的代码：

```
<style type="text/css">
        #main
        {
            position: absolute;
            width: 700px;
            height: 180px;
            top: 50%;
            left: 50%;
            margin-top: –90px;
            margin-left: –350px;
        }
 </style>
```

查看 id 为“main”的 div 发生了什么变化。

注：上边的样式目的是为了使登录框在任何类型的浏览器中，登录框都位于浏览器的中间位置。水平居中比较方便设置，但是设置垂直居中比较麻烦。

（3）切换到“源”视图，在 id 为“main”的 div 内添加如下代码。

```
<fieldset></fieldset>              // <fieldset>标记用来创建圆角矩形
```

（4）在“源”视图中，选中“fieldset”标记，打开“应用样式”窗口，单击“新建样式”选项，在弹出的“新建样式”对话框中设置“选择器”为“级联样式”，选择“定位”类别，设置其 width 为“100%”，height 为“100%”，修改后的代码如下：

```
<fieldset style="width: 100%; height: 100%">
</fieldset>
```

（5）切换到“设计”视图，把光标定位到“fieldset”内，选择“表”→“插入表”菜单项，在弹出的“插入表格”对话框中设置“行数”为“1”，“列数”为“2”，“宽度和高度”为“100%”，单击“确定”按钮，在 fieldset 标记内插入一个 1 行 2 列的表格。

（6）选中左边列，打开“应用样式”窗口，单击“新建样式”按钮，在弹出的“新建样式”对话框中设置“选择器”为“级联样式”。选择“定位”类别，设置其 width 为

“350 px”；选择“边框”类别，设置右边框的 border-style 为“solid”；选择“背景”类别，设置背景图片为“主题”文件夹内 image 文件夹中准备好的图片“system.gif”，background-repeat 为“no-repeat”，x 和 y 轴的 background-position 为“center”。以同样的方法选中右边列，设置其 width 为“350 px”，设置完毕后效果如图 5.10 所示。

图 5.10　左部效果图

（7）把光标定位到第二个 td 标记内，选择“表”→“插入表”菜单项，在弹出的“插入表格”对话框中设置“行数”为“4”，“列数”为“2”，宽度为“100%”，单击“确定”按钮，在此列中插入一个 4 行 2 列的表格。

（8）打开“应用样式”窗口，单击“新建样式”按钮，在弹出的“新建样式”对话框中设置“选择器”为“.right”，“定位位置”为“当前页面”。选择“定位”类别，设置 width 属性为“150 px”，height 属性为“35 px”；选择“块”，设置 text-align 属性为“right”。单击“确定”按钮，在“当前页”内创建一个嵌入式的 css 样式规则，代码如下：

```
.right
{
        width: 150px;
        height: 35px;
        text-align: right;
}
```

（9）打开“应用样式”窗口，单击“新建样式”按钮，在弹出的“新建样式”对话框中设置“选择器”为“.left”，“定位位置”为“当前页面”。选择“定位”类别，设置 width 属性为“200 px”，height 属性为“35 px”；选择“块”类别，设置 text-align 属性为“left”。单击“确定”按钮，在“当前页”内创建一个嵌入式的 css 样式规则，代码如下：

```
.left
{
        width: 200px;
        height: 35px;
        text-align: left;
}
```

（10）选中右边单元格内的表格的第一列，打开“应用样式”窗口，选中“right”样式，给第一列应用“right”样式规则。选中右边单元格内的表格的第二列，打开“应用样式”窗口，选中“left”样式，给第二列应用“left”样式规则。

（11）如图 5.11 所示，在表格内添加相应的文本和控件（标准选项卡内），并按照表 5.3 设置相应控件的属性。

图 5.11　右部效果图

表 5.3　控件属性设置

控　　件	设　　置
第一行的文本框	ID 为“TextBoxUserName”，Width 为“150 px”
第二行的文本框	ID 为“TextBoxUserPassword”，Width 为“150 px”，TextMode 为“Password”
第三行的下拉列表框	ID 为“DropDownListUserType”，Width 为“150 px”，Items 中添加 4 个成员，每个成员的 Text 和 Value 属性分别为：“===选择用户列别===”，“null”；“管理员”，“0”；“教师”，“1”；“学生”，“2”
第四行的按钮	ID 为“loginButton”，Text 为“登录”，Width 为“80 px”；ID 为“ButStuRegister”，Text 为“学生注册”，Width 为“80 px”

（12）右击此页面，选择“在浏览器中查看”选项，浏览器中的效果如图 5.12 所示。

图 5.12　用户登录

3．编写系统登录代码

（1）编写所需的基础类库

为了密码管理的完全性，不能直接保存用户的明文密码，而应该把用户的密码通过 MD5 加密后保存到数据库中。此学生成绩管理系统的教师密码由管理员提供，所以这里不对教师密码加密，而只对学生密码进行加密。因为这个方法经常使用，所以把其放入一个类中，方便调用，其中还可以放置一些经常使用的方法。右击 App_Code 文件夹，打开“添加新项”对话框，选择“类”模板，修改名称为“commTools.cs”，类内部的代码如下：

```
public class commTools
{
    public commTools()
    {          }
    //用于在浏览器端弹出提示框的一段代码
    public static void alert(Page page, string msg)
```

```
        {
            string scriptString = "alert('" + msg + "')";
            ScriptManager.RegisterClientScriptBlock(page, typeof(Page),
            DateTime.Now.ToString().Replace(":", " "), scriptString, true);
        }
        //根据 RadioButtonList 值设置其某项被选中
        public static void setRadioButtonListByValue(RadioButtonList list,string value)
        {
            for(int i = 0; i < list.Items.Count;i++ )
            {
                if (list.Items[i].Value == value)
                    list.Items[i].Selected = true;
                else
                    list.Items[i].Selected = false;
            }
        }
        public static string toMD5(string str)              //MD5 加密
        {
            return System.Web.Security.FormsAuthentication.HashPasswordForStoringInConfigFile(str,
             "md5");
        }
    }
```

（2）编写登录验证代码

当用户在登录框中填写完用户信息，单击“登录”按钮后，系统会利用用户的输入信息在系统数据库中进行查询，判断用户是否在系统中存在，如果存在，继续判断用户的密码是否正确，并根据判断的情况给用户提示不同的信息。双击“登录”按钮，切换到 Default.aspx.cs 代码页，系统自动添加了 loginButton_Click 方法，引入命名空间如下：

```
using System.Collections.Generic;
using System.Data.SqlClient;
```

添加 loginButton_Click 方法中的代码如下：

```
if (this.DropDownListUserType.SelectedValue != "null")
{
    //获取用户输入的信息
    string userType = this.DropDownListUserType.SelectedValue.ToString();
    string userName = this.TextBoxUserName.Text;
    string userPassword = this.TextBoxUserPassword.Text;
    //定义用来判断查询结果的变量
    Boolean isExist = false;
    string sql = "";
    //定义保存操作参数的集合
```

```
List<SqlParameter> cmdpara = new List<SqlParameter>();
if (userType == "2")      //学生登录
{
    //利用学生编号在学生表内查询，判断是否存在此学生
    sql = "SELECT stuID FROM Student WHERE stuID=@stuID";
    //初始化相关变量
    isExist = false;
    cmdpara.Clear();
    //向集合内添加需要的参数
    cmdpara.Add(commDBHelper.CreateParameters("@stuID", DbType.String, userName));
    commDBHelper.ExecuteReader(sql, CommandType.Text, out isExist, cmdpara);
    if (isExist)
    {
        //存在此用户，继续判断密码是否正确
        sql = "SELECT stuID,stuName FROM Student WHERE stuID=@stuID and
        stuPassword=@stuPassword";
        //初始化相关变量
        isExist = false;
        cmdpara.Clear();
        //向集合内添加需要的参数
        cmdpara.Add(commDBHelper.CreateParameters("@stuName",DbType.String, serName));
        cmdpara.Add(commDBHelper.CreateParameters("@stuPassword", DbType.String,
            commTools.toMD5(userPassword)));
        DataRow row = commDBHelper.ExecuteReader(sql, CommandType.Text, out isExist,
            cmdpara);
        if (isExist)
        {
            //用户名和密码正确，则利用 Sessiion 对象保存用户的状态
            Session["userName"] = row["stuName"];
            Session["userID"] = row["stuID"];
            Session["userType"] = "2";
            Session["userLogin"] = true;
            //利用 Response.Redirect 方法调转到用户子系统中的某页面
            Response.Redirect("~/students/StuSearchScore.aspx ");
         }
         else
        {
            //密码输入问题
            commTools.alert(this, "密码不对，请确认，如果忘记请联系管理员！ ");
        }
    }
```

```
        else
        {
            //此用户不存在
            commTools.alert(this, "此用户不存在，请确认！");
        }
    }
    else if (userType == "0" || userType == "1")      //教师登录
    {
        //教师或者管理员，利用教师编号和教师类型在数据库内查询，判断是否存在此用户
        sql = "SELECT teaID FROM Teacher    WHERE teaName=@teaName and teaType=@teaType";
        //初始化相关变量
        isExist = false;
        cmdpara.Clear();
        //向集合内添加需要的参数
        cmdpara.Add(commDBHelper.CreateParameters("@teaName", DbType.String, userName));
        cmdpara.Add(commDBHelper.CreateParameters("@teaType", DbType.String, userType));
        commDBHelper.ExecuteReader(sql, CommandType.Text, out isExist, cmdpara);
        if (isExist)
        {
            //存在此用户，继续判断密码是否正确
            sql = "SELECT teaName,teaID FROM Teacher WHERE teaName=@teaName AND
                teaPassword=@teaPassword and teaType=@teaType";
            //初始化相关变量
            isExist = false;
            cmdpara.Clear();
            //向集合内添加需要的参数
            cmdpara.Add(commDBHelper.CreateParameters("@teaName",DbType.String,userName));
            cmdpara.Add(commDBHelper.CreateParameters("@teaPassword", DbType.String,
                userPassword));
            cmdpara.Add(commDBHelper.CreateParameters("@teaType", DbType.String, userType));
            DataRow row = commDBHelper.ExecuteReader(sql, CommandType.Text, out isExist,
                cmdpara);
            if (isExist)
            {
                //用户名和密码正确，则利用 Session 保存用户的状态
                Session["teaID"] = row["teaID"];
                Session["userType"] = userType;
                Session["userLogin"] = true;
                //利用 Response.Redirect 方法跳转到用户子系统中的某页面
                if (userType == "0")
                {
```

```
                    Response.Redirect("~/Admin/teacher.aspx");
                }
                if (userType == "1")
                {    Response.Redirect("~/Teacher/enterScore.aspx");    }
            }
            else
            {
                //密码输入问题
                commTools.alert(this, "密码不对，请确认，如果忘记请联系管理员！");
            }
        }
        else
        {
            //此用户不存在
            commTools.alert(this, "此用户不存在，请确认！");
        }
    }
}
else
{
    //没有选择用户类别
    commTools.alert(this, "请先选择用户类别！");
}
```

（3）编写用户登录检测代码

当用户登录系统后，如果没有退出，就可以在所允许的范围内任意浏览页面，无须重新登录。这个功能主要利用 Session 会话对象中保存的登录用户的信息自动进行判断，来确定用户是否能够访问所访问的页面。

当他们访问一个页面时，Page_load 事件首先会被执行，因此可以在页面的 Page_load 事件内从 Session 会话对象中取出用户的登录信息，利用Session 中的用户信息判断用户是否有权限访问所要访问的页面。如果有权限，则会在浏览器中显示出用户所请求的页面，如果用户没有权限访问，则会自动在后台跳转到用户登录页面，让用户进行登录，所访问的页面就无法显示出来。

此部分的代码会被所有需要权限控制的页面使用到，因此把这部分代码封装成一个方法，放在 commTools 类中，在需要安全访问控制的页面的 Page_load()方法内调用此方法即可，这样方便代码的修改和升级。封装在 commTools 类内的代码如下：

```
/// <summary>
/// 判断用户是否能够访问响应的页面，如果不能访问，则进行页面跳转
/// </summary>
/// <param name="page">所要判断的页面对象</param>
/// <param name="userType">指定能够访问的用户类型</param>
```

```
public static void checkUser(Page page,string userType)
{
    //从 session 中获取当前登录的用户的信息
    string currentUserType = (string)page.Session["userType"];
    Boolean userLogin = (Boolean)page.Session["userLogin"];
    //如果用户还没有登录就访问此页面，或者非允许的用户类型访问此页面就跳转到登录页面
    if (userLogin==false || (currentUserType != userType))
    {          page.Response.Redirect("~/Default.aspx"); }
}
```

（4）添加“学生注册”按钮功能代码

当单击“学生注册”按钮后链接到注册页面（StudentRegister.aspx）。双击“学生注册”按钮，在“ButStuRegister_Click”方法中添加如下代码：

```
Response.Redirect("~/Students/StudentRegister.aspx");
```

此系统的权限要求比较简单，对应的用户类型只能访问对应类型的页面，没有复杂的权限模型，因此这里的用户登录检测方法比较简单，在实际的项目中需要根据系统的权限模型进行增强。

第二部分　知识点链接

L5.1　学生信息注册

L1. Web控件概述

控件是一种类，绝大多数控件都具有可视的界面，能够在程序运行中显示外观。利用控件进行可视化设计既直观又方便，可以实现“所见即所得”的效果。程序设计的主要内容是选择和设置控件，以及对控件的事件编写处理代码。

ASP.NET 中的控件被组织成两个名称空间，即 System.Web.UI.HtmlControls 和 System.Web.UI.WebControls。System.Web.UI.HtmlControls 名称空间包含 HTML 服务器控件，该类服务器控件直接映射到 HTML 元素上；System.Web.UI.WebControls 名称空间包含 Web 服务器控件，Web 服务器控件更丰富、抽象。与 ASP.NET 1.1 版本相比，ASP.NET 2.0 增加了近 60 个控件，ASP.NET 3.5 又增加了很多控件，从而大大提高了快速开发的能力。

网页控件的层次结构如图 5.13 所示。

由于 ASP.NET 提供的每一个控件均有众多的属性、方法和事件，本书在讲解 HTML 服务器控件和 Web 服务器控件时仅选取常用且重要的内容详细介绍，需要查阅较全面的有关控件属性、方法和事件列表的读者，可以参阅附录 B 的内容或微软提供的 MSDN 帮助文档。

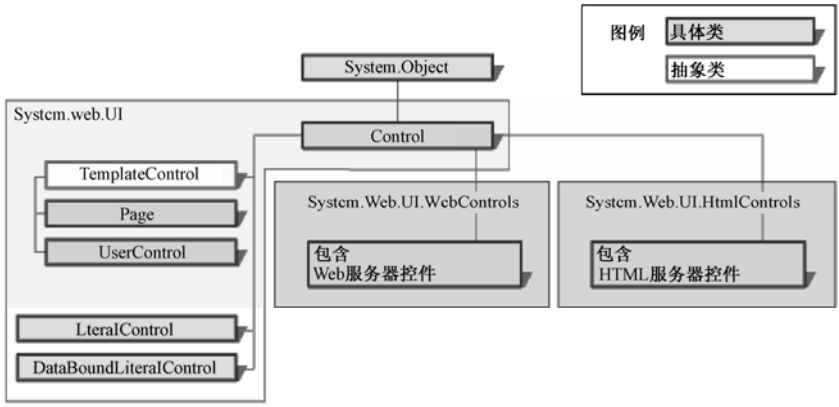


图 5.13　ASP.NET 网页控件的层次结构

L2. HTML服务器控件

1. HTML服务器控件的层次结构

HTML 服务器控件是特殊的 HTML 元素，这些元素包含使其自身在服务器上可见并可编程的属性。默认情况下，服务器无法使用 Web 窗体页上的 HTML 元素，这些元素被视为传递给浏览器的不透明文本。但是，通过将 HTML 元素转换为 HTML 服务器控件，可将其公开为可在服务器上编程的元素。

Web 窗体页上的任意 HTML 元素都可以转换为 HTML 服务器控件。转换是一个只涉及几个属性的简单过程。作为最低要求，通过添加 RUNAT="SERVER"属性，HTML 元素即可转换为控件。如果要在代码中作为成员引用 HTML 服务器控件，还应当为控件分配 ID 属性。图 5.14 显示了 HTML 服务器控件的层次结构。

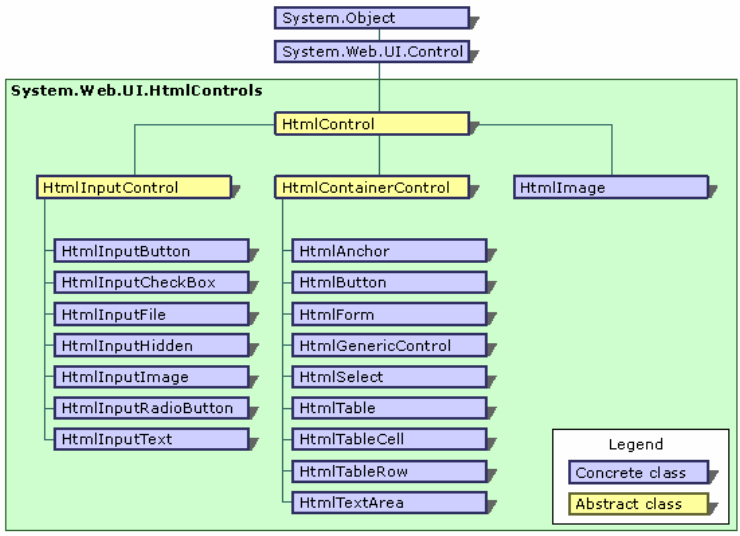


图 5.14　HTML 服务器控件的层次结构

2. HTML服务器控件的基本语法

定义 HTML 服务器控件的基本语法是：

```
<HTML 标记 id= "控件名称" runat="Server">
```

由于 HTML 服务器控件是由 HTML 标记所衍生出来的新功能，因此在所有的 HTML 服务器控件的语法中，最前端是 HTML 标记，不同的控件用不同的标志来标记；runat= "Server" 表示控件将会在服务器端执行；id 用来设置控件的名称，在同一程序中各控件的 id 均不相同，id 属性允许以编程方式引用该控件。

表 5.4 列举了 HTML 标签，以及它们所属的类别。

表 5.4　HTML 标签及其类别

HTML 标签	类 别	HTML 服务器控件名称	说　　明
<head>	容器	HtmlHead	<head>元素。可以在它的控件集合中添加其他元素
<input>	输入	HtmlInputButton	<input type=button | submit | reset>
		HtmlInputCheckbox	<input type=checkbox>
		HtmlInputFile	<input type=file>
		HtmlInputHidden	<input type=hidden>
		HtmlInputImage	<input type=image>
		HtmlInputPassword	<input type=password>
		HtmlInputRadioButton	<input type=radio>
		HtmlInputReset	<input type=reset>
		HtmlInputSubmit	<input type=submit>
		HtmlInputText	<input type=text | password>
<img>	空	HtmlImage	图片
<link>	空	HtmlLink	Href 属性读取/设置 URL 目标
<textarea>	容器	HtmlTextArea	多行文本输入
<a>	容器	HtmlAnchor	锚标签
<button>	容器	HtmlButton	服务器端按钮，可自定义显示格式，IE 4.0 及以上版本可用
<form>	容器	HtmlForm	每个页面最多只能有一个 HtmlForm 控件；method 默认为 POST
<table>	容器	HtmlTable	表格，可以包含行，行中包含单元格
<td> <th>	容器	HtmlTableCell	表格单元格；表格标题单元格
<tr>	容器	HtmlTableRow	表格行
<title>	容器	HtmlTitle	标题元素
<select>	容器	HtmlSelect	用于选择的下拉菜单

实际上，在一个内容文件中，如页面、用户控件或母版页中，不会仅使用表 5.4 所示的 HTML 服务器控件名称。在 HTML 代码中，一般都会包括 runat="server" 属性和 id 属性。

服务器不会处理普通的 HTML 控件，如<h1>、<a>和<input>，它们将直接被发送到客户端，由浏览器进行显示。如果要让 HTML 控件能在服务器端被处理，就要将它们转换为 HTML 服务器控件。

将普通 HTML 控件转换为 HTML 服务器控件，只需简单地添加 runat="server"属性。另外，可能还需要添加 id 属性，这样可以通过编程方式访问和控制控件。例如，下面是一个简单的输入控件：

```
<input type="text" size="40">
```

可以添加 id 和 runat 属性，将它转换为 HTML 服务器控件，如下：

```
<input type="text" id="BookTitle" size="40" runat="server">
```

因为 HTML 控件只能运行在客户端，而不是服务器端，所有往往在 ASP.NET 下运行现有的 HTML 页时需进行这样的转换。转换后的 HTML 控件具有服务器控件的特点：可以使用面向对象技术对其进行编程控制；提供了一组事件，可以为事件编写事件处理程序；自动维护控件；允许自定义属性等。

在表 5.4 所列出的控件中，有几个共同的属性会经常被使用，如 Innerhtml、InnerText、Disabled、Visible、Value、Attributes 及 Style。下面分别进行介绍。

（1）InnerHtml 属性：获取或设置控件的开始标记和结束标记之间的内容，但不自动将特殊字符转换为等效的 HTML 实体。

下例说明如何使用 InnerHtml 属性动态设置文本消息。

```
<HTML>
<SCRIPT Language="C#" Runat="Server">
protected void Page_Load(object sender, EventArgs e)
{
    Message.InnerHtml = "Welcome! You accessed this page at: " + DateTime.Now.ToString();
}
</SCRIPT>
<BODY>
    <SPAN   Id="Message" Runat=server></SPAN>
</BODY>
</HTML>
```

（2）InnerText 属性：获取或设置控件的开始标记和结束标记之间的内容，并自动将特殊字符转换为等效的 HTML 实体。

（3）Value 属性：用来获取各种输入字段的值，包括 HtmlSelect、HtmlInputText 等。

（4）Attributes 属性：服务器控件标记上表示的所有属性名称和值的集合，使用该属性可以用编程方式访问 HTML 服务器控件的所有特性。

L3. Web服务器控件

在 ASP.NET 的工具箱中，只有 HTML 选项卡中的控件是浏览器端控件，其他各种控件都是服务器控件。其中“标准”选项卡中的控件是较常用的控件。在类库中，所有的网页控件都是从 System.Web.UI.Control.WebControls 直接或间接派生而来的。

服务器控件包含方法及与之关联的事件处理程序，并且这些代码都在服务器端执行（部分服务器控件也提供客户端脚本，尽管如此，这些控件事件仍然会在服务器端处理）。

如果控件包括可视化组成部分（如标签、按钮和表格），那么 ASP.NET 将在检测目标浏览器接收能力的情况下，为浏览器呈现传统的 HTML。如果 ASP.NET 服务器控件需要利用客户端脚本以实现某些功能，如本章所描述的验证控件那样，那么将会生成适应于浏览器类型的脚本，并发送给浏览器。然而，服务器端验证过程仍然执行。

需要特别注意的是，发送给客户端的永远是最普通的 HTML 代码，所以，ASP.NET 应用程序可以运行在任何厂商的任何浏览器上。所有处理过程都在服务器端完成，且

ASP.NET 服务器控件最终呈现在浏览器中的是标准的 HTML 代码。另外，所发送的脚本并非是必须经过优化的。

ASP.NET 服务器控件提供统一的编程模型。例如，在 HTML 中，input 标签（<input>）可用于按钮、单行文本域、复选框、隐藏域和密码。而多行文本域，则必须使用<textarea>标签。使用 ASP.NET 服务器控件时，每种不同的功能类型都将对应一种特定控件。例如，使用 TextBox 控件输入文本，并通过属性指定行数。通常情况下，对于 ASP.NET 服务器控件而言，所有声明标记的属性都与控件类的属性相对应。

1. Web服务器控件的层次结构

所有呈现到浏览器的、具有可视化外观的 Web 服务器控件，都从 WebControl 类派生。该类提供了所有 ASP.NET 服务器控件的通用属性、方法和事件。其中包括常用属性，如 BorderColor、BorderStyle 和 BorderWidth，以及 RenderBeginTag 和 RenderEndTag 方法。

WebControl 类和其他一些 ASP.NET 服务器控件（如 Literal、PlaceHolder、Repeater 和 XML）是从 System.Web.UI.Control 派生，而 System.Web.UI.Control 又从 System.Object 派生。Control 类提供了一些基本属性，如 ID、EnableViewState、Parent 和 Visible，以及一些基本方法，如 Dispose、Focus 和 RenderControl，还包括一些生命周期事件，如 Init、Load、PreRender 和 Unload。

从 Control 类派生的 WebControl 类和控件，位于 System.Web.UI.WebControls 命名空间中。它们之间的关系如图 5.15 所示。

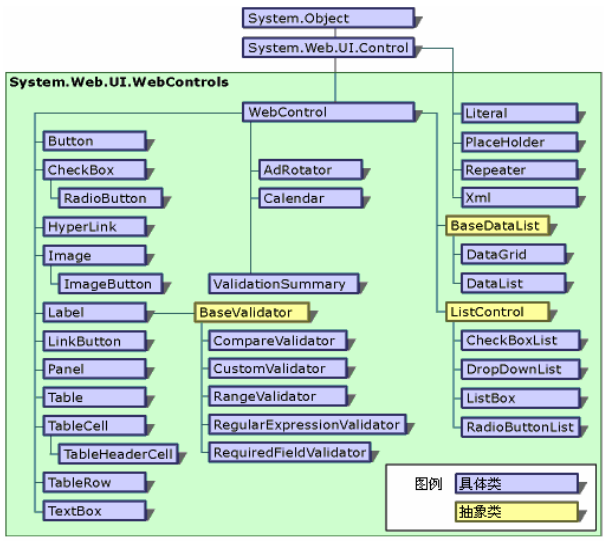


图 5.15　Web 服务器控件的层次结构

Web 服务器控件继承了 WebControl 和 System.Web.UI.Control 类的所有属性、事件和方法。表 5.5 列出了从 Control 或 WebControl 类继承的 Web 服务器控件的常用属性。

表 5.5　Web 服务器控件的常用属性

名 称	类　型	值	说　明
AccessKey	String	单字符的字符串	按【Alt】键加上它的值，可以使控件得到焦点
BackColor	Color	Azure、Green、Blue 等	背景颜色

续表

名 称	类　型	值	说　明
BorderColor	Color	Fuchsia、Aqua、Coral 等	边框颜色
BorderStyle	BorderStyle	Dashed、Dotted、Double、NotSet 等	边框样式。默认为 NotSet
BorderWidth	Unit	nn、nnpt	边框的宽度。如果用 nn，nn 是整数，单位是像素。如果用 nnpt，nn 是整数，单位是点
CausesValidation	Boolean	True、False	表示是否输入控件引发控件所需的验证。默认值为 True
Controls	ControlCollection		该控件所包含的所有控件对象的集合
CssClass	String		CSS 类
Enabled	Boolean	True、False	若设为 False，则控件可见，但显示为灰色，不能操作。内容仍可复制和粘贴。默认值为 True
EnableViewState	Boolean	True、False	表示该控件是否维持视图状态。默认值为 True
Font	FontInfo		
ForeColor	Color	Lavender、LightBlue、Blue 等	前景色
ID	String		控件的可编程标识符
Parent	Control	页面上的控件	返回在页面控件层次结构中对该控件的父控件的引用
EnableTheming	Boolean	True、False	表示是否将主题应用到该控件
SkinID	String	皮肤文件名	应用到该控件的主题文件夹下的皮肤文件的详细信息
ToolTip	String		当鼠标移动到控件上方的时候显示出的文本字符串，在低版本的浏览器中呈现
Visible	Boolean	True、False	若设 False，则不呈现该控件。默认值为 True

2. Web服务器控件基本语法

ASP.NET 服务器控件的基本语法为：

```
<asp:controlType id="ControlID" runat="server" thisProperty="this value" thatProperty="that value"/>
```

控件标签总是以“asp:”开头，这就是标记前缀。controlType 是控件的类型或类，如 Button、CheckBoxList、GridView 等。可以利用 id 属性，以编程方式引用控件实例。runat 属性告知服务器，该控件在服务器端运行。

注：不能认为 runat="server" 是默认属性就无须在每个控件中都声明它，而必须在每个控件的每次声明中都显式地包括该属性。如果省略了它，并不会发生错误，但是控件将被忽略而不被呈现。如果省略 id 属性，控件能完全呈现出来，但是该控件无法在代码中引用和操作。

可以在尖括号中声明其他属性。例如，为 TextBox 声明 Text 和 Width 属性，如下所示：

```
<asp:TextBox id="txtBookName" runat="server" Width="250px" Text="Enter a book name.">
</asp:TextBox>
```

虽然 ASP.NET 允许标记的属性值可以不加引号，但是鉴于 ASP.NET 服务器控件必须使用良构的 XHTML 语法，不建议读者这样做。需要明确的是，如上例所示，一般情况下，标签是成对出现的，也就是由起始标签<asp:TextBox>和结束标签</asp:TextBox>构成。但

若此标签仅占一行，也可在标签最后加一个“/”作为结束。所以，上面的 TextBox 也可以书写为：

```
<asp:TextBox id="txtBookName" runat="server" Width="250px" Text="Enter a book name."/>
```

另外，许多 Web 服务器控件可以在起始和结束标签之间使用内部 HTML。例如，在 TextBox 控件中，可将 Text 属性指定为内部 HTML，而不是将其设置在打开标签的属性中。所以，上面的控件又可以等价地写为：

```
<asp:TextBox id="txtBookName" runat="server" Width="250px">Enter a book name.</asp:TextBox>
```

3. Web服务器控件使用详解

（1）TextBox 文本框控件

TextBox 文本框控件是用得最多的控件之一，该控件可以用来显示数据或输入数据。定义的格式如下：

```
<asp:Textbox id="TextBox1" Text = "请在这里输入数据" Column="25" MaxLengh="35" runat="server" />
```

TextBox 控件有一个重要的属性：TextMode。该属性包括 3 个选项：

① SingleLine：单行编辑框。

② MultiLine：带滚动条的多行文本框。

③ PassWord：密码输入框，所有输入字符都用特殊字符（如“*”）来显示。

许多浏览器都支持自动完成功能，该功能可帮助用户根据以前输入的值向文本框中填充信息。自动完成的精确程度取决于浏览器。通常，浏览器根据文本框的 name 属性存储值；任何同名的文本框（即使是在不同页上）都将为用户提供相同的值。TextBox 控件支持 AutoCompleteType 属性，该属性提供了用于控制浏览器如何使用自动完成的选项。

除此以外，TextBox 控件还有一个常用的 TextChanged 事件，当文字改变时引发此事件，可以编写事件处理代码做出响应。

下面的代码演示如何响应 TextBox 控件中的更改，在标签中显示文本框控件的内容。

```
protected void TextBox1_TextChanged(object sender, EventArgs e)
{
    Label1.Text = Server.HtmlEncode(TextBox1.Text);
}
```

默认情况下，TextChanged 事件并不马上导致向服务器回发 Web 页，而是当下次发送窗体时在服务器代码中引发此事件。若要使 TextChanged 事件引发即时发送，需将 TextBox 控件的 AutoPostBack（自动回传）属性设置为 True。

（2）Button、LinkButton 和 ImageButton 按钮控件

网页控件中的按钮分为三种：Button、LinkButton 和 ImageButton。它们功能基本相同，但外观上有区别。Button 的外观与传统按钮的外观相同；LinkButton 的外观与超链接字符串相同；ImageButton 按钮用图形方式显示外观，其图像通过 ImageURL 属性来设置。

三种按钮的功能都与 HTML 的提交按钮（Submit Button）相似，即每当这些按钮被单击（Click）时，就将缓冲区中的事件信息一并提交给服务器。

定义上述三种按钮的语法如下：

```
<asp:Button id="Button1" runat="server" Text="按钮"></asp:Button>
<asp:LinkButton id="LinkButton1" runat="server">链接按钮</asp:LinkButton>
<asp:ImageButton id="ImageButton1" runat="server" ImageUrl="…"></asp;ImageButton>
```

下面的代码功能是用鼠标单击 LinkButton 按钮，该按钮即可通过服务器转向新的网页，从而起到“超链接”的作用。在 LinkButton 按钮的 Click 事件中写出以下程序：

```
private void LinkButton1_Click(object sender, System.EventArgs e)
{
    Response.Redirect("其他窗体的 URL");
}
```

三种按钮的共同属性：

① PostBackUrl 属性：利用这个属性可以将按钮变成“返回”按钮。即先将该属性设成某个网页的 URL，以后单击该按钮时就会直接转向该网页。

② OnClientClick 属性：定义当单击按钮时执行的客户端脚本，通常是一个脚本函数的函数名。

③ CommandName 属性：当在网页上具有多个按钮控件时，可使用 CommandName 属性来指定或确定与每一个按钮控件关联的命令名。可以用标识要执行的命令的任何字符串来设置 CommandName 属性。然后，以编程方式确定按钮控件的命令名并执行相应的操作。

三种按钮的共同事件：

① Click 事件：按钮的 OnClick 属性对应的值就是此事件添加的处理函数的函数名。处理函数将在服务器端执行，如果为某个按钮控件同时指定了 OnClientClick 属性和 OnClick 属性，那么将优先响应客户端的处理。

② Command 事件：当单击按钮控件时会引发 Command 事件。通常当命令名（如 Sort）与按钮控件关联时，才会使用该事件。这使得可以在一个网页上创建多个按钮控件，并以编程方式确定单击了哪个按钮控件。

当用户单击按钮控件时，该页回发到服务器。默认情况下，该页回发到其本身，重新生成相同的页面并处理该页上控件的事件处理程序。

默认情况下，表单以 Post 方法提交页面。Button 控件支持 Post，但 LinkButton 和 ImageButton 控件却不直接支持 Post。使用后两种类型的按钮时，它们将客户端脚本添加到页面以允许控件以编程方式提交页面。因此 LinkButton 和 ImageButton 控件要求在浏览器上启用客户端脚本。

（3）CheckBox 和 CheckBoxList 复选控件

CheckBox 和 CheckBoxList 复选控件为用户提供了在真/假、是/否或开/关选项之间进行选择的方法。CheckBox 是单个控件，可以单独使用。而 CheckBoxList 控件是复选框列表控件，可以将多个 CheckBox 组合在一起使用。使用单个 CheckBox 控件时，更容易控制页面上的布局。例如，可以在各个复选框之间包含文本，也可以单独控制复选框的字体和颜色。如果想用数据库中的数据创建一系列复选框，则 CheckBoxList 控件是较好的选择。

使用 CheckBoxList 时要给控件增添选项。方法是先选择该控件，然后找到控件的 Items 属性，单击右边的省略号按钮，将弹出如图 5.16 所示的对话框。

利用“添加”按钮，按照图 5.16 中的选项得出的页面部分如图 5.17 所示。

图 5.16　编辑 CheckBoxList 的选项

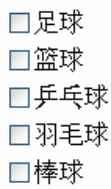


图 5.17　CheckBoxList 选项

HTML 控件中的 CheckBox 控件只能用于静态处理，而 Web 服务器控件中的 CheckBox 可以进行数据绑定操作，所以通常用于动态数据绑定。关于数据绑定，将在后续章节中介绍。

对于 CheckBox 控件，常用的事件是 CheckedChanged，单个 CheckBox 控件在用户单击该控件时引发 CheckedChanged 事件。默认情况下，这一事件并不导致向服务器发送页面，但通过将 AutoPostBack 属性设置为 True，可以使该控件强制立即发送。需要说明的是，不论 CheckBox 控件是否发送到服务器，都可以不必为 CheckedChanged 事件创建事件处理程序。直接在处理程序中测试选中了哪个复选框更为方便。通常，只有在更关心复选框的状态更改而不是读取其值时，才会为 CheckedChanged 事件创建一个事件处理程序。

（4）RadioButton 和 RadioButtonList 单选控件

RadioButton 和 RadioButtonList 单选控件的作用和使用方法与 CheckBox 基本相同，唯一的差别在于，在一个 RadioButtonList 内的多个 RadioButton 之间只能有一项被选中，而在 CheckBoxList 中可以同时选中多项。

（5）ListBox 控件

ListBox 控件通常用于一次显示一个以上的项。可以在以下两方面控制列表的外观：

① 显示的项数。可将该控件设置为显示特定的项数。如果该控件包含比设置的项数更多的项，则显示一个垂直滚动条。

② 高度和宽度。可以以像素为单位设置控件的大小。在这种情况下，控件将忽略已设置的行数，而是显示足够多的行直至填满控件的高度。

ListBox 控件的主要属性和事件如下：

① AutoPostBack 属性：若该属性为 True，则当更改选项内容后会自动回发到服务器；若为 False，则不回发。

② Rows 属性：表示可以显示的选项行数。

③ Items 属性：是 ListBox 控件各选项的集合。通过该属性可以获取对当前存储在 ListBox 中的项列表的引用，通过此引用，可以在集合中添加项、移除项和获得项的计数。每个列表项都是一个单独的对象，具有自己的属性，见表 5.6。

表 5.6　ListBox 控件中列表项的基本属性

属　性	说　明
Text	列表中显示的文本
Value	与某个项关联的值。设置此属性可将该值与特定的项关联，而不显示该值。例如，可以将 Text 属性设置为某个职员的名字，将 Value 属性设置为该职员的电子邮件别名
Selected	布尔值，指示该项是否被选定。如果 ListBox 被设置为允许多重选择，则可选择一项以上

④ SelectionMode 属性：该属性指明一次是否可多选。SelectionMode 属性可以有两个取值：Single 表明在 ListBox 中仅能选择一项；Multiple 表明可以选择多项。如果将 ListBox 控件设置为允许进行多重选择，则可以在按住【Ctrl】或【Shift】键的同时，单击选择多个项。

当用户单击列表选项时，ListBox 控件将引发 SelectedIndexChanged 事件。默认情况下，此事件不会导致将页发送到服务器，可以通过将 AutoPostBack 属性设置为 True，使此控件强制立即回发。下面的代码示例演示如何响应 ListBox 控件中的选择。事件处理程序将用户选择的选项显示在 Label 控件中。

```
Protected void ListBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    Label1.Text = "You selected " + ListBox1.SelectedItem.Text;
}
```

（6）DropDownList 控件

DropDownList 控件是一个用下拉框方式显示选项的控件，它允许用户从预定义下拉列表中进行单行选择。该控件与 ListBox 控件的不同之处在于：它在框中显示选定项，同时还显示下拉按钮。若用户单击此按钮，将显示项的列表；另外，DropDownList 控件不支持多重选择模式。

DropDownList 控件使用语法：

```
<ASP: DropDownList 属性设置 > </ASP: DropDownList >
```

或

```
<ASP: DropDownList 属性设置 />
```

DropDownList 控件的主要属性和事件如下：

① AutoPostBack 属性：若该属性为 True，则当更改选项内容后会自动回发到服务器；若为 False，则不回发。

② Items 属性：Items 属性是 DropDownList 控件各选项的集合。每个列表项都是一个单独的对象，具有自己的属性，同 ListBox 控件。

DropDownList 控件可用来列出从某个数据源读取的选项。DropDownList 控件中的每一项分别对应数据源中的一项（通常是一行）。相关属性和事件如下：

① DataSource 属性：获取或设置此 DropDownList 控件的数据源。

② DataTextField 属性：指明用于提供选项文本的数据源字段。

③ DataValueField 属性：该属性表示指定数据源的相关数据字段的值。

④ SelectedIndexChanged 事件：当用户选择一项时，DropDownList 控件将引发 SelectedIndexChanged 事件。默认情况下，此事件不会导致向服务器发送页面，但当将

AutoPostBack 属性设置为 True 时，此控件强制立即发送。

（7）HyperLink 超链接控件

HyperLink 控件提供了一种使用服务器代码在 Web 页上创建和操作链接的方法，使用户可以在应用程序中的页之间移动。

HyperLink 超链接控件有 4 个重要的属性。

① Text 属性：HyperLink 控件的文本标题。

② ImageUrl 属性：使用 ImageUrl 属性指定为 HyperLink 控件显示的图像。HyperLink 控件可以显示为文本或图像，如果同时设置了 Text 和 ImageUrl 属性，则 ImageUrl 属性优先。如果图像不可用，则显示 Text 属性中的文本。

③ NavigateUrl 属性：使用 NavigateUrl 属性指定单击 HyperLink 控件时定位到的 URL。

④ Target 属性：使用 Target 属性指定单击 HyperLink 控件时显示链接到哪个 Web 窗口或页框架。

使用 HyperLink 控件而不是传统的 HTML 超链接标签的好处是：

① 可以在服务器代码中设置链接属性。例如，可以基于页面中的条件动态地更改链接文本或目标页。

② 可以使用数据绑定来指定链接的目标 URL（以及必要时与链接一起传递的参数）。

需要注意的一点，与大多数 Web 服务器控件不同的是，当用户单击 HyperLink 控件时并不会在服务器代码中引发任何事件，此控件只用于导航。

（8）Image 与 ImageMap 图像控件

利用 Image 图像控件可以在 Web 窗体页上显示图像，并用自己的代码来管理这些图像。图像源文件可以在设计时确定，也可以在程序运行中指定，还可以将控件的 ImageURL 属性绑定到数据源上，根据数据源的信息来选择图像。

与大多数其他 Web 服务器控件不同，Image 控件不支持鼠标单击（click）事件。如果需要使用鼠标单击事件时，可以使用 ImageButton 控件来代替 Image 控件。

显示一个图像所需的最少操作是：先创建一个 Image 控件，然后指定一个图像文件。具体步骤如下。

进入“设计”视图，在“工具箱”中展开“标准”选项卡，然后将一个 Image 控件拖放到网页界面上。将控件的 ImageURL 属性设置为.gif、.jpg 或其他网络图形文件的 URL。给 Image 控件设置以下属性。

① Height 和 Width：在页面上为图形保留适当空间（高度和宽度）。

② ImageAlign：用来设置图像对齐的方式。可使用的值包括 Top、Bottom、Left、Middle 和 Right。

③ AlternateText：有的浏览器不支持加载图像时，替代图像的文本。

ImageMap 控件可以用来显示图像，也可以实现图像的超链接。该控件的最大特点是，可以将 ImageMap 中的图像按照（X,Y）坐标划分成不同形状的区域，分别链接到不同的网页。该控件的 ImageUrl 属性用来连接图像源文件；HotSpot 属性用来划分链接区域。单击 HotSpot 属性右边的省略号按钮，弹出如图 5.18 所示的对话框。

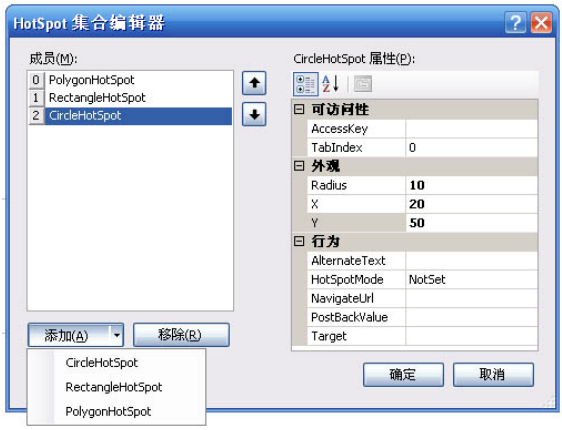


图 5.18　编辑 ImageMap 控件的 HotSpot

利用“添加”按钮的下拉列表可以选择区域的形状，在右边的属性栏目中可以确定区域的位置及链接的网页。

（9）FileUpload 控件

应用程序中经常需要允许用户把文件上传到 Web 服务器。尽管在 ASP.NET 1.X 也可以完成该功能，但在 ASP.NET 中使用 FileUpload 控件会更简单。

该控件让用户更容易地浏览和选择用于上传的文件，它包含一个浏览按钮和用于输入文件名的文本框。只要用户在文本框中输入了完全限定的文件名，无论是直接输入或通过浏览按钮选择，都可以通过调用 FileUpload 的 SaveAs 方法保存到磁盘上。

用户选择要上传的文件后，FileUpload 控件不会自动将该文件保存到服务器。必须显式提供一个控件或机制，使用户能提交指定的文件。例如，可以提供一个按钮，用户单击它即可上传文件。为保存指定文件所写的代码应调用 SaveAs 方法，该方法将文件内容保存到服务器上的指定路径。通常，在引发回发到服务器的事件的事件处理方法中调用 SaveAs 方法。例如，如果提供一个用于提交文件的按钮，则可以将执行文件保存操作的代码放在单击事件的事件处理方法中。

可以通过 FileUpload 的一些重要属性访问上传的文件：

① FileName 属性：用来获取客户端上使用 FileUpload 控件上传文件的名称。此属性返回的文件名不包含此文件在客户端上的路径。

② FileBytes 属性：该属性从使用 FileUpload 控件指定的文件返回一个字节数组，包含了指定文件的内容。

③ FileContent 属性：该属性获取 Stream 对象，该对象指向使用 FileUpload 控件上传的文件。可以使用 FileContent 属性来访问文件的内容。例如，可以使用该属性返回的 Stream 对象以字节方式读取文件内容，并将它们存储在一个字节数组中。

④ PostedFile 属性：PostedFile 属性获取文件的基础 HttpPostedFile 对象（该文件使用 FileUpload 控件上传）。使用该属性还可访问上传文件的其他属性。例如，可以使用 ContentLength 属性来获取文件的长度，使用 ContentType 属性来获取文件的 MIME 内容类型。

需要注意的是，防止攻击者利用 FileUpload 控件发起拒绝服务攻击的方法之一就是限

制使用 FileUpload 控件上传文件的大小。应当根据要上传的文件的类型，设置与类型相适应的大小限制。默认大小限制为 4096KB（4 MB）。可以通过设置 web.config 配置文件中 httpRuntime 元素的 maxRequestLength 属性设置允许上传更大的文件。

（10）Panel 控件

Panel Web 服务器控件在 Web 窗体页内提供了一种容器控件，可以将它用做静态文本和其他控件的父级。此外，可将 Panel 控件用于其他目的：

① 对控件和标记进行分组。对于一组控件和相关的标记，可以通过把其放置在 Panel 控件中，然后操作此 Panel 控件，以将它们作为一个单元进行管理。例如，可以通过设置 Panel 的 Visible 属性来隐藏或显示该面板中的一组控件，而不必去设置面板中每一个控件的 Visible 属性。

② 具有默认按钮的窗体。可将 TextBox 控件和 Button 控件放置在 Panel 控件中，然后通过将 Panel 控件的 DefaultButton 属性设置为面板中某个按钮的 ID，来定义一个默认的按钮。如果用户在面板内的文本框中进行输入时按 Enter 键，这将与单击特定的默认按钮具有相同的效果。

③ 动态生成的控件的容器。Panel 控件为在运行时创建的控件提供了一个方便的容器。

Panel 控件有以下常用属性：

① HorizontalAlign：指定子控件在面板内的对齐方式（左对齐、右对齐、居中或两端对齐）。

② Direction：指定控件的内容是从左到右还是从右到左进行呈现。当在页面上创建与整个页面的方向不同的区域时，此属性非常有用。

③ ScrollBars：可以通过设置此属性来添加水平或垂直方向的滚动条。

（11）Calendar 控件

Calendar 服务器控件在 ASP.NET 网页中显示一个单月份日历。用户可使用该日历查看和选择日期。该控件基于.NET Framework DataTime 对象，因此支持该对象所允许的全部日期范围。用户可有效地显示从公元 0 年～9999 年之间的任意日期。

使用该控件的一般语法形式如下：

```
<asp:Calender ID="MyCalendar" runat="server"></asp:Calendar>
```

Calender 控件的常用属性如表 5.7 所示。

表 5.7　Calender 控件的常用属性

属 性 名	说　明
SelectedDate	被选中的日期
SelectionMode	用户被允许选择日期的方式。可使用下列值之一：None(无)、Day(日)、DayWeek(日、星期)、DayWeekMonth(日、星期、月)
SelectionDayStyle	被选中日的格式
TadayDayStyle	当天日期的样式

（12）MultiView 与 View 控件

View 控件和 MultiView 控件可以作为控件或标记的容器，一般情况下，可以用 View 控

件实现一个页面的功能，而 MultiView 控件可以操纵多个 View 控件。

View 控件与 MultiView 控件作为控件或标记的容器，提供了一种很方便地切换信息视图的方式。使用 View 控件与 MultiView 控件可以实现如下的任务。

① 根据用户的选择或其他条件提供不同的页面。例如，可能允许用户从一个列表中选择，其中每个项都有独立的 View 控件与之对应，然后可以显示用户选择的 View 控件内的内容。

② 创建多页窗体。View 控件与 MultiView 控件可以提供与 Wizard 控件类似的行为。例如，每个操作可能需要多个步骤才可以完成，不同的步骤所要显示的界面不同，就可以在 MultiView 内放多个 View，在不同的 View 内绘制不同的界面，通过用户的操作步骤，依次显示出对应的视图。

MultiView 控件用做一个或多个 View 控件的外部容器，View 控件可以包含标记和控件的任何集合。

MultiView 控件一次只能显示一个 View 控件，并显示此 View 控件内的标记和控件。通过重置 MultiView 控件的 ActiveViewIndex 属性，可以指定当前可见的 View 控件。

通过 MultiView 控件的 ActiveViewIndex 属性可以设置要显示的 View 控件，其中第一个 View 为 0，第二个为 1，依次类推。MultiView 控件还支持添加到每个 View 内部的导航控件。若要创建导航按钮，可以向每个 View 控件添加一个按钮控件，然后可以将每个按钮的 CommandName 和 CommandArgument 属性设置为空字符串，以使 MultiView 控件切换到另一个 View。表 5.8 列出了 CommandName 值和相应的 CommandArgument 值的组合。

表 5.8　不同的命令类型与命令参数

CommandName 值	CommandArgument 值
NextView	不需要设置
PrivView	不需要设置
SwitchViewByID	要切换到的 View 控件的 ID
SwitchViewByIndex	要切换到的 View 控件的索引号

L4. Web验证控件

验证工作最好放在客户端进行。当在客户端输入完数据，向服务器提交之前应对数据进行检测，如果发现错误，立即提示并要求改正，改正前不向服务器提交信息。这种处理方式可以将改正错误的过程放在提交以前，减少网上的无效传输。

但是有两个原因使客户端验证不可依赖。第一，由于相当一部分客户端的设备功能弱，不具备验证能力，此时验证工作只能放在服务器端进行；第二，恶意的用户能够比较容易地破坏客户端的验证脚本，或者想方设法绕过客户端的校验。因此，从安全的角度出发，不论客户端是否进行了验证，服务器端的验证都是不可缺少的，除非人为地取消了服务器端验证。当用户向服务器提交数据之后，服务器都毫无例外地调用验证程序来逐个检查用户的输入。如果发现任何输入数据有错误，整个页面将自行设置为无效状态，并发出错误信息。

1. 验证控件的分类及作用

ASP.NET 提供了 6 种验证控件。各种验证控件的作用如下：

① RequiredFieldValidator 控件：用于检查用户是否在输入控件中输入了数据。

② CompareValidator 控件：将输入控件的值与常数值或其他输入控件的值相比较，以确定这两个值是否与由比较运算符（小于、等于、大于）指定的关系相匹配。

③ RangeValidator 控件：用于检查用户的输入是否在一个特定的范围内。

④ RegularExpressionValidator 控件：用于检查用户的输入是否与正则表达式所定义的模式匹配。

⑤ CustomValidator 控件：通过用户自定义的验证函数判定输入的数据是否有效。

⑥ ValidationSummary 控件：以列表的形式显示页面上所有验证控件所搜索到的验证错误。

其中 ValidationSummary 控件只能与前 5 种验证控件一起使用，不能单独执行验证。另外在这些控件中，除 RequiredFieldValidator 控件以外，其他所有的控件都认为空字段是合法的。

2. 使用验证控件

各个控件虽然作用不同，但使用的方法却有很多共同点。除了 ValidationSummary 控件外，其他的验证控件都继承于共同的基类 BaseValidator，每个控件都有一个 ControlToValidate 属性，必须用它来指定被验证的控件名称。下面分别介绍各验证控件的使用方法。

（1）RequiredFieldValidator 控件

RequiredFieldValidator 控件用于对一些必须输入的信息进行检验，如果一些必须输入的数据没有输入，将提示错误。

使用这个控件的方法比较简单，将控件拖入窗体以后，关键是给它设置以下 4 个属性：

① ControlToValidate：设置被验证的控件，可以在该属性的下拉列表中选择。

② ErrorMessage：当不能通过验证时显示的错误信息。

③ Display：显示错误信息的位置，包括以下 3 种选择。

a）None：不显示错误信息。

b）Static：显示在设计时控件所放置的位置。

c）Dynamic：将错误信息动态显示在页面上。

④ EnableClientScript：该属性为逻辑变量，默认为 True，表示如有可能（如浏览器版本为 Internet Explorer 4.0 以上），先在客户端验证。若将本属性值改为 False，将不在客户端进行验证。

（2）CompareValidator 控件

CompareValidator 控件用来将输入到控件（如 TextBox 控件）的值与输入到其他控件的值或常数值进行比较。几个重要的属性的设置方法如下：

① 通过设置 ControlToValidate 属性指定被验证的控件 ID。

② 如果要将输入控件与其他控件值进行比较，将 ControlToCompare 属性设置为要与之相比较的控件 ID。如果要将输入控件的值与某个常数值进行比较，应将 ValueToCompare 属性设置为与之比较的常数。

③ Type 属性用于设置比较数据的类型，只有在同一类型的数据之间才能够进行比较。

④ Operator 属性用来指定比较的方法，如大于、等于。如果将 Operator 属性设置为 Validation-CompareOperator.DataTypeCheck，则 CompareValidator 控件将忽略 ControlTo-Compare 和 ValueToCompare 属性，并且仅仅指示输入到输入控件中的值是否可以转换为 BaseCompare-Validator.Type 属性指定的数据类型。

（3）RangeValidator 控件

RangeValidator 控件用于检查用户的输入是否在一个特定的范围内，可以检查数字对、字母对和日期对限定的范围，边界表示为常数。

RangeValidator 控件的主要属性包括：

① ControlToValidate：指定被验证的输入控件。

② MinimumValue 和 MaximumValue：分别指定有效范围的最小值和最大值。

③ Type：用于指定要比较的值的数据类型。

（4）RegularExpressionValidator 控件

控件 RegularExpressionValidator 控件用来验证输入的格式是否匹配某种特定的模式（正则表达式）。这类验证允许检查一些可以预知的字符序列，如身份证号码、电子邮件地址、电话号码和邮编中的字符序列等。除非浏览器不支持客户端验证，或者已明确禁止客户端验证（通过将 EnableClientScript 属性设置为 False），否则将同时执行服务器端和客户端验证。

客户端的正则表达式验证实现与服务器端的略有不同。在客户端，使用的是 JScript 正则表达式语法。而在服务器端，使用的是 System.Text.RegularExpressions.Regex 语法。由于 JScript 正则表达式语法是 System.Text.RegularExpressions.Regex 语法的子集，所以最好使用 JScript 正则表达式语法，以便在客户端和服务器端得到同样的结果。

使用本控件进行校验时，除按照前面几个控件设置属性以外，最主要的区别是将控件的 ValidationExpress 属性设置检查模式。方法是单击属性右边的省略号按钮，在弹出的对话框中选择“标准表达式”，弹出的对话框如图 5.19 所示，然后选择需要检查的模式即可。

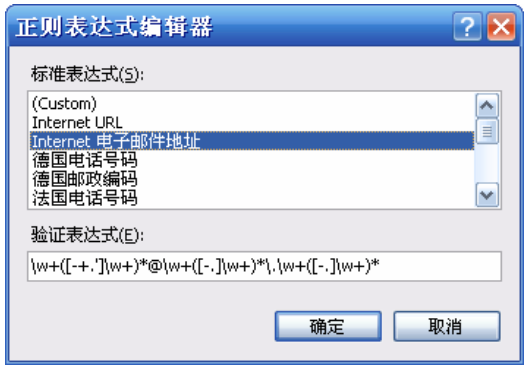


图 5.19　正则表达式编辑器

注：在编写处理字符串的程序或网页时，经常会有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些规则的工具。换句话说，正则表达式就是记录文本规则的代码。正则表达式具有复杂的语法定义，感兴趣的读者可以参阅：http://unibetter.com/deerchao/zhengzhe-biaodashi-jiaocheng-se.htm，网络上有很多常用字符规则的验证表达式，可以到网上参考。

（5）CustomValidator 控件

使用自定义控件 CustomValidator 时，可以自行定义验证算法，并同时利用控件提供的其他功能。

为了在服务器端验证函数，先将 CustomValidator 控件拖入窗体，并将 ControlToValidate 属性指向被验证的对象，然后给该验证控件的 ServerValidate 事件提供一个验证程序，最后在 ErrorMessage 属性中填写出现错误时显示的信息。

在 ServerValidate 事件处理程序中，可以从 ServerValidateEventArgs 参数的 Value 属性中获取输入到被验证控件中的字符串。验证的结果要存储到 ServerValidateEventArgs 的属性 IsValid（true 或 false）中。

例如，利用自定义 CustomValidator 控件验证某个输入框输入的数据能否被 3 整除。若不能被 3 整除，发出错误信息。事件处理的代码如下：

```
private void CustomValidator1_ServerValidate(object source,
                                    System.Web.UI.WebControls.ServerValidateEventArgs args)
{
    int number=int.Parse(args.Value);              // 取出输入的数据
    if((number % 3) == 0)                          // 校验能否被 3 整除
        args.IsValid=true;                         // 结果正确
    else
        args.IsValid=false;                        // 结果错误
}
```

如果需要同时提供客户端验证程序以便让具有 DHTML 能力的浏览器先进行验证，应该在.aspx 的 HTML 视图中用 JavaScript 语言编写验证程序，同时将验证的函数名写入控件的 ClientValidationFunction 属性中。

（6）ValidationSummary 控件

ValidationSummary 控件用于在一个位置上集中显示来自 Web 网页上所有验证程序的错误信息。根据 DisplayMode 属性的设置，可以采用列表、项目符号列表或单个段落的形式来显示。通过设置控件的 ShowSummary 和 ShowMessageBox 属性，可以确定显示的形式。

L5.2 系统登录

L1. 内置对象概述

ASP.NET 定义了多个内置对象，它们是全局对象，即不必事先声明就可以直接使用。例如，Response.Write("Hello")，就是直接使用了 Response 对象，传送信息到浏览器。

每个对象有各自的属性、（属性）集合、方法和事件。属性用来描述对象的性质，表示对象的静态特性；方法反映对象的行为，表示对象的动态特性；集合指一组相关的值，如 Request 对象的 QueryString 由一组相关值构成；事件指对象在一定条件下产生的信息，如对于 Session 对象，会话开始将产生 OnStart 事件。访问对象属性的格式如下。

```
对象名.属性名
```

例如，访问 Page 对象的 IsPostBack 属性的语法格式为：Page.IsPostBack。访问对象的集合与访问对象属性类似，对象的集合都有一个 count 属性，它表示集合中值的个数。

访问对象方法的格式如下。

```
对象名.方法名(参数表)
```

例如，访问 Response 对象的 Write 方法的语法格式为 Response. Write("Hello");。

对象事件处理的定义如下。

```
对象名_事件名(参数表)
事件名(参数表)
```

例如：

```
protected void Page_Load(object sender, EventArgs e)
{
    //程序体定义
}
```

ASP.NET 事件处理过程都有以下两个参数：

① object sender：事件处理过程的第一个参数，表示发生该事件的源对象；

② EventArgs e：事件处理过程的第二个参数，表示传递给事件处理过程的额外描述，作为辅助之用。

ASP.NET 内置的对象主要有 7 个，如表 5.9 所示。

表 5.9　ASP.NET 的内置对象

对 象 名	说　明
Page	用于设置与网页有关的属性、方法和事件
Request	从浏览器（用户端）获取信息
Response	发送信息到浏览器
Server	提供服务器端的属性和方法
Session	存储单个客户端的信息
Application	存储客户端的共享信息
Exception	捕捉 ASP.NET 的错误，返回错误描述

除了内置对象外，ASP.NET 还包含其他的对象，如文件类和数据库类对象，它们是数据存储和访问的主要手段，需在对象创建以后使用。

L2. Response

Response 对象的主要作用是输出数据到客户端。Response 对象类名是 HttpResponse，它是 Page 对象的成员，可直接使用。

Response 对象提供了许多属性和方法，常用属性如表 5.10 所示。

表 5.10　Response 对象的常用属性

属　性	说　明	类　型
BufferOutput	设定 HTTP 输出是否启用缓冲处理，预设为 True	Boolean
Cache	传回目前网页缓存的设置	HttpCachePolicy
Charset	设置或取得 HTTP 的输出字符编码	String
Cookies	传回目前请求的 HttpCookieCollection 对象集合	HttpCookieCollection
IsClientConnected	传回客户端是否仍然处于和 Server 连接中	Boolean
StatusCode	传回或设定输出至客户端浏览器的 HTTP 状态码，预设是 200	Integer
StatusDescription	传回或设定输出至客户端浏览器的 HTTP 状态说明字符串	String
SuppressContent	设定是否将 HTTP 的内容送至客户端浏览器，若为 True，则网页将不会传至 Client 端	Boolean

Response 对象的常用方法如表 5.11 所示。

表 5.11　Response 对象的常用方法

方　法	说　明
AppendToLog	将自定的记录信息加到 IIS 的日志文件中
BinaryWrite	将一个二进制的字符字符串写入 HTTP 输出串流
Clear	将缓冲区的内容清除
ClearHeaders	将缓冲区中所有的页面标头清除
Close	关闭客户端的联机
End	将目前缓冲区中所有的内容送到客户端，然后关闭联机
Flush	将缓冲区中所有的数据送到客户端
Redirect	将网页重新导向另一个地址
Write	将数据输出到客户端
WriteFile	将一个文件直接输出至客户端

1. 地址重定向

Response 对象的 Redirect 方法可以将连接重新导向到其他地址，使用时只要传入一个字符串型的 URL 即可，传入在网址后附加参数的 URL 字符串也可以。例如：

```
Response.Redirect("http://www.baidu.com");
```

2. 直接输出文本文件

Response 对象提供了一个直接输出文本文件的 WriteFile 方法。若所要输出的文件和执行的网页在同一个文件夹，只要直接传入文件名就可以了；若不在同一个文件夹，则要指定详细的路径。

例如，将与本程序位于同一文件夹的文本文件 Output.txt 直接输出到网页上，代码如下：

```
Response.WriteFile("Output.txt");
```

3. 检查使用者的联机状态

当网页在执行需要较长时间的复杂运算或循环时，使用者的浏览器会一直处于等待的状态。此时若使用者停止浏览的动作，而服务器还继续执行运算的话，将浪费有限的系统资源。因此可以在执行这些需要等待的运算时，判断 Response 对象的 IsClientConnected 属性：若为 False 则代表使用者已经离线，此时只要使用 Response 对象的 End 方法来结束网页的执行即可。这样 Server 就不会执行无用的工作，而可以空出更多的资源来让他人使用。

【例 5-1】 设计一个页面，检测用户是否在线。

设计步骤如下所示：

（1）运行 Visual Studio 2008，新建一个命名为“InnerObject”的 ASP.NET 网站，添加一个命名为“DetectingIsOnline.aspx”的 Web 窗体文件。

（2）在 DetectingIsOnline.aspx 的 Page_Load 方法中输入如下代码。

程序代码执行可达 100 000 次的循环，在循环中加入对 IsClientConnection 属性的判断操作；只要 Client 端离线就终止执行，而 Client 端的浏览器上也不会出现任何信息。

```
public void Page_Load(object Sender, Eventargs e)
{
    long i;
    for (i = 0; i <= 100000; i++)
    {
        if (!Response.IsClientConnected)
            Response.End();
    }
    Response.Write("执行完毕");
}
```

（3）在浏览器中查看本页面，服务器会在后台运行那个耗时的循环，页面不会立刻显示出来，但是如果关闭本页面，服务器就不会再处理此运算。

L3. Request

Request 对象的主要功能是把客户端浏览器的数据传递给服务器。与 Response 对象一样，Request 对象也是 Page 对象的成员之一，所以在程序中也不需要做任何的说明即可直接使用。Request 对象所属的类名是 HttpRequest。Request 对象的属性相当多，表 5.12 列出了常用的属性。

表 5.12　Request 对象的常用属性

属　性	说　明	类　型
ApplicationPath	传回目前正在执行程序的服务器端的虚拟目录	String
Browser	传回有关客户端浏览器的功能信息	HttpBrowserCapabilities
ClientCertificate	传回有关客户端安全认证的信息	HttpClientCertificate
ContentEncoding	传回客户端所支持的字符设定。中文 Internet Explorer 预设是 ChineseTraditional(Big5)	Encoding

续表

属　性	说　明	类　型
ContentType	传回目前需求的 MIME 内容类型	String
Cookies	传回 HttpCookieCollection 对象集合	HttpCookieCollection
FilePath	传回目前执行网页的相对地址	String
Form	传回有关窗体变量的集合	NameValueCollection
Headers	传回有关 HTTP 标头的集合	NameValueCollection
HttpMethod	传回目前客户端 HTTP 数据传输的方式是 Post 或 Get	String
IsAuthenticated	传回目前 HTTP 联机是否有效	Boolean
IsSecureConnection	传回目前 HTTP 联机是否安全	Boolean
Params	传回 QueryString、Form、ServerVariable 及 Cookies 全部的集合	NameValueCollection
PhysicalApplicationPath	传回目前执行的 Server 端程序在 Server 端的真实路径	String
PhysicalPath	传回目前请求网页在 Server 端的真实路径	String
QueryString	传回附在网址后面的参数内容	NameValueCollection
RawUrl	传回目前请求页面的原始 URL	String
RequestType	传回客户端 HTTP 数据的传输方式使用 Get 或 Post	String
ServerVariables	传回网页 Server 变量的集合	NameValueCollection
Url	传回有关目前请求的 URL 信息	HttpUrl
UserAgent	传回客户端浏览器的版本信息	String
UserHostAddress	传回远方客户端机器的主机 IP 地址	String
UserHostName	传回远方客户端机器的 DNS 名称	String
UserLanguages	传回一个储存客户端机器使用的语言	String

Request 对象的常用方法有以下两个：

（1）MapPath(virtualPath)：将参数 virtualPath 指定的虚拟路径转化为实际路径。

（2）SaveAs(filename,includeHeaders)：将 HTTP 请求保存到磁盘，filename 是保存的文件路径，includeHeaders 指定是否保存 HTTP 标头。

1. 读取表单数据

Request 对象可以用来捕获由客户端返回服务器端的数据，如读取表单的数据，或读取保存在用户本地计算机上的 Cookie 等。

读取表单数据的方式有以下三种（与表单数据返回服务器的方式有关）。

（1）使用 Request 的 QueryString 属性获取表单数据。如果在 HtmlForm 控件中将 Method 属性设置为“Get”，则表单数据将以字符串形式附加在网址的后面返回服务器端，此时必须使用 Request 的 QueryString 属性来捕获表单数据。

Get 方式可以传递多个值给 URL 所指定的页面。

传递参数的格式：URL?var1=value&var2=value…

（2）使用 Request 的 Form 属性获取表单数据。如果将 Method 属性设置为“Post”，则表单数据将以放在 HTTP 标头的形式传到服务器端，此时需要使用 Request 对象的 Form 属性来捕获表单数据。

（3）使用 Request 对象的 Param 属性来获取表单数据。无论 Method 是“Get”还是

“Post”，均可以使用 Request 对象的 Param 属性来获取表单数据。

在使用时还可以省略 QueryString、Form、Param 属性来得到表单的数据。以下是使用 Request 对象获取表单数据的例子：

```
Request.Querystrng("username");     //以 Get 方法传送表单数据，服务器端得到表单项 username 的值
Request.Form("username");           //以 Post 方法传送表单数据，服务器端得到表单项 username 的值
Request.Param("username");          //服务器端得到表单项 username 的值
Request("username");                //省略属性，得到表单项 username 的值
```

2. 取得客户端浏览器的信息

使用 Request 对象的 Browser 属性可取得目前连接 Web 服务器的浏览器的信息。Browser 属性是一个集合对象，所以可使用一个 HttpBrowserCapabilities 型态的对象变量来接收 Browser 属性的传回值。

【例 5-2】 设计一个页面来获取客户端浏览器的相关信息。

设计步骤如下。

（1）运行 Visual Studio 2008，打开例 5-1 命名为“InnerObject”的 ASP.NET 网站，添加命名为“DetectingClientInfo.aspx”的 Web 窗体文件。

（2）在 DetectingClientInfo.aspx 的 Page 的 load 事件中加入如下代码，使用 HttpBrowserCapabilities 型态的变量来取得浏览器的部分信息。

```
protected void Page_Load(object Sender, EventArgs e)
{
    HttpBrowserCapabilities bc = Request.Browser;
    Response.Write("<P>浏览器信息:</P>");
    Response.Write("浏览器 = " + bc.Browser + "<BR>");
    Response.Write("名称 = " + bc.Browser + "<BR>");
    Response.Write("版本 = " + bc.Version + "<BR>");
    Response.Write("使用平台 = " + bc.Platform + "<BR>");
    Response.Write("是否为测试版 = " + bc.Beta + "<BR>");
    Response.Write("是否为 16 位的环境 = " + bc.Win16 + "<BR>");
    Response.Write("是否为 32 位的环境 = " + bc.Win32 + "<BR>");
    Response.Write("是否支持框架(Frame) = " + bc.Frames + "<BR>");
    Response.Write("是否支持表格(Table) = " + bc.Tables + "<BR>");
    Response.Write("是否支持 Cookie = " + bc.Cookies + "<BR>");
    Response.Write("是否支持 VB Script = " + bc.VBScript + "<BR>");
    Response.Write("是否支持 Java Script = " + bc.JavaScript + "<BR>");
    Response.Write("是否支持 Java Applets = " + bc.JavaApplets + "<BR>");
    Response.Write("是否支持 ActiveX Controls = " + bc.ActiveXControls + "<BR>");
}
```

（3）在浏览器（IE 7.0）中查看本页，运行结果如图 5.20 所示。

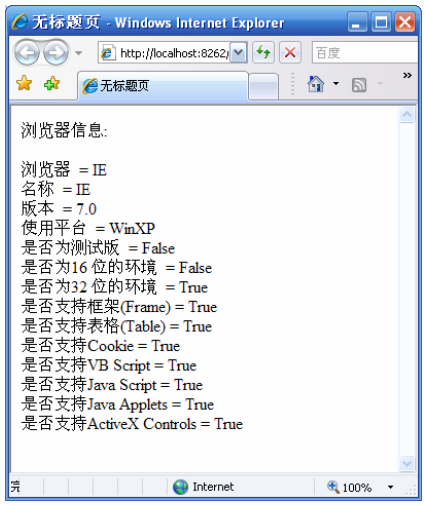


图 5.20　获取浏览器信息

L4. Application

Application 对象的用途是记录整个网站的信息。Application 对象可以用来记录不同浏览器端共享的变量，无论有几个浏览者同时访问网页，都只会产生一个 Application 对象，也就是所有正在使用这个网页程序的浏览器端都可以存取这个变量。Application 对象变量的生命周期起始于 Web 服务器开始执行时，终止于 Web 服务器关机或重新启动时。

Application 对象的类名称是 HttpApplication，每个 Application 变量都是 Application 对象集合中的对象之一，由 Application 对象统一管理。

语法：

```
Application["变量"] = "变量内容";
Application["对象名"] = Server.CreateObject[ProgId];
```

上述语法用于创建 Appication 变量，所产生的变量可以使用 Application 对象的属性与方法。Application 对象是 Page 对象的成员，可以直接使用。

表 5.13、表 5.14 分别列出了 Application 对象的常用属性和方法。

表 5.13　Application 对象的常用属性

属　性	说　明	类　型
All	传回全部的 Application 对象变量到一个 Object 类型的数组	Object
AllKeys	传回全部的 Application 对象变量名称到一个 String 类型的数组中	String
Count	取得 Application 对象变量的数量	Integer
Item	通过索引或 Application 变量名称传回内容值	Object

表 5.14　Application 对象的常用方法

方　法	说　明
Add	新增一个新的 Application 对象变量
Clear	清除全部的 Application 对象变量
Get	使用索引值或变量名称传回变量值
GetKey	使用索引值来取得变量名称

续表

方　法	说　　明
Remove	使用变量名称移除一个 Application
RemoveAll	移除全部的 Application 对象变量
Set	使用变量名称更新一个 Application 对象变量的内容
Lock	锁定全部的 Application 变量
UnLock	解除锁定 Application 变量

1. 存取Application对象变量值

取得和设置 Application 对象变量值的语法如下。

```
变量 = Application["变量名称"];        Application["变量名称"] = 表达式;
```

例如，Application["count"] = 0。

【例5-3】 利用 Application 对象存储变量，向 Application 对象中添加 3 个变量并赋值，3 个变量的值分别为“Value1”、“Value2”和“Value3”。通过循环显示这 3 个 Application 变量的名字和值，显示完以后清除它们。

设计步骤如下所示：

（1）运行 Visual Studio 2008，打开例 5-1 命名为“InnerObject”的 ASP.NET 网站，新建命名为“AccessApplication.aspx”的 Web 窗体文件。

（2）在 AccessApplication.aspx 的 Page_Load 方法中加入如下代码：

```
    protected void Page_Load(object Sender, EventArgs e)
    {
        short shtI;
        //向 Application 对象中添加 3 个变量并赋值
        Application.Add("App1", "Value1");
        Application.Add("App2", "Value2");
        Application.Add("App3", "Value3");
        for (shtI = 0; shtI <= Application.Count - 1; shtI++)
        {
            //显示 3 个 Application 变量名和值
            Response.Write("变量名：" + Application.GetKey(shtI));
            Response.Write(" ,变量值：" + Application[shtI] + "<P>");
        }
        //清除 Application 对象变量
        Application.Clear();
}
```

注：上述程序利用 Application 的 Add()方法，向 Application 对象中添加 3 个值分别为“Value1”、“Value2”和“Value3”的变量。所有的 Application 变量都放在 Application 集合中，由 Application 对象管理。所以要取出集合中的对象，可使用 for 循环或 foreach 循环，循环中分别使用 GetKey 方法传回变量名称，Item 属性传回变量内容，程序最后将集合中的变量用 Clear 方法清除。

（3）在浏览器中查看本页，程序运行结果如图 5.21 所示。

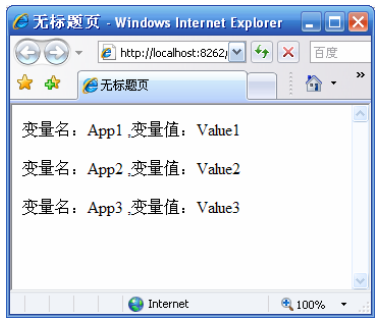


图 5.21　Application 变量的存取

2. 锁定Application对象

因为 Application 对象变量是共享的，每个连线的客户端都可以使用，所以可能造成多个使用者同时存取同一变量的情形，从而导致存入的数据不正确。要避免这种情况，可利用 Application 对象的 Lock 方法将变量暂时锁定，禁止他人写入，等操作完毕后再利用 Application 对象的 UnLock 方法解除锁定。

语法：

```
Application.Lock();
Application["变量"] = 表达式;
Application.UnLock();
```

第 1 条语句锁定 Application 变量，第 2 条语句改变指定的 Application 变量值，第三条语句取消对 Application 变量的锁定。

3. Application对象的事件

Application 事件只能在 Global.asax 文件中定义，Global.asax 文件必须存放在 Web 主目录中。当浏览器与 Web 服务器连接时，会先去检查 Web 主目录有没有 Global.asax 文件，如果有，则先执行该文件。

Application 对象有以下 4 个事件：

（1）OnStart 事件：在整个 ASP.NET 应用中首先被触发的事件，也就是在一个虚拟目录中第一个 ASP.NET 程序执行时触发。

（2）OnEnd 事件：与 OnStart 正好相反，在整个应用停止时被触发（通常发生在服务器被重启/关机时）。

（3）OnBeginRequest 事件：在每一个 ASP.NET 程序被请求时发生，即客户每访问一个 ASP.NET 程序时，就触发一次该事件。

（4）OnEndRequest 事件：ASP.NET 程序结束时，触发该事件。

以下是一个 Global.asax 文件的代码示例。

```
<%@ Application Language="C#" %>
<script runat="server">
    void Application_Start(object sender, EventArgs e)
    {
```

```
        //在应用程序启动时运行的代码
        Application.Add("count", 0);
    }
    void Application_End(object sender, EventArgs e)
    {
        //在应用程序关闭时运行的代码
        Application.RemoveAll();
    }
</script>
```

L5. Session

Session 对象的功能和 Application 对象一样，都是用来记录浏览器端的变量，但两者的差异是：Application 对象记录的是所有浏览器端共享的变量，而 Session 对象变量只记录单个浏览器端专用的变量。也就是说各个在线的机器有各自的 Session 对象变量，但共享同一个 Application 对象。Session 对象的生命周期起始于浏览器第一次与服务器连接时，终止于浏览器结束连接或浏览器超过 Timeout 属性设置的分钟数没有访问网页。

Session 对象类名称是 HttpSessionState，它和 Application 对象一样属于 Page 对象的成员，所以可以直接使用。Session 对象的使用方式和 Application 对象变量很相似。

语法：

```
Session["变量名"] = 表达式;
Session["对象名称"] = Server.CreateObject(ProgId);
```

例如：

```
Session["name"]="李明";
```

表 5.15、表 5.16 分别列出了 Session 对象的常用属性和方法。

表 5.15　Session 对象的常用属性

属　性	说　明	类　型
All	传回全部的 Session 对象变量到一个数组	Object
Count	传回 Session 对象变量的个数	Integer
Item	以索引值或变量名称来传回或设定 Session 对象变量的内容	Object
TimeOut	传回或设定 Session 对象变量的有效时间，当联机使用者超过有效时间没有动作，Session 对象便失效。默认值为 20 分钟	Integer
SessionID	由服务器生成的会话 ID，它代表唯一的一次会话	Integer

表 5.16　Session 对象的常用方法

方　法	说　明
Add	新增一个 Session 对象变量
Clear	清除所有的 Session 对象变量
Remove	以变量名称来移除变量
RemoveAll	清除所有的 Session 对象变量
Abandon	结束 Session 对象，目前的 Session 对象会失效

1. 设定Session对象变量的有效期限

因为每一个和服务器端联机的客户端都有独立的 Session，所以服务器端需要额外的资源来管理这些 Session。有时使用者正在浏览网页时，可能去做其他的事情而没有把网页的联机关闭。如果服务器端一直浪费资源在管理这些 Session 上，那么势必会降低服务器的效率。所以当使用者超过一段时间没有动作时，就可以将 Session 释放。要更改 Session 对象的有效期限，只要设定 TimeOut 属性即可。TimeOut 属性的默认值是 20 分钟。

【例 5-4】 将 Session 对象的 TimeOut 属性设定为 1 分钟，在程序中加入 var1 和 var2 两个变量，它们的值分别为“Value1”和“Value2”。程序运行时将显示两个变量的值，超过 1 分钟后，单击“显示”按钮，再查看两个变量的值，此时这两个变量的值将消失。

设计步骤如下所示：

（1）运行 Visual Studio 2008，打开例 5-1 命名为“InnerObject”的 ASP.NET 网站，新建命名为“SessionTimeOut.aspx”的 Web 窗体文件。

（2）在 SessionTimeOut.aspx 可视化设计界面中放 3 个标签控件 Label1、Label2 和 Label3，分别用来显示当前时间和两个 Session 变量的值。分别在 Label2 和 Label3 的前面输入“第一个 Session 的值：”和“第二个 Session 的值：”；放一个按钮控件 Button1，其 Text 属性设置为“显示”。

（3）在 SessionTimeOut.aspx 的 Page_Load 方法中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        Session["var1"] = "Value1";
        //添加一个 Session 变量 Session1
        Session["var2"] = "Value2";
        //添加另一个 Session 变量 Session2
        Session.Timeout = 1;
        //设置 Session 变量有效期为 1 分钟
        Label1.Text = DateTime.Now.ToLongTimeString();
        //设置时间的显示格式
        Label2.Text = Session["var1"].ToString();
        //显示变量 Session1 的值
        Label3.Text = Session["var2"].ToString();
    }
}
```

（4）在设计视图中双击 Button1 按钮，在 Button1 的 Click 事件内添加如下代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
    Label1.Text = DateTime.Now.ToLongTimeString();
    Label2.Text = Session["Session1"] == null ?
                                    string.Empty : Session["Session1"].ToString();
```

```
        Label3.Text = Session["Session2"] == null ?
                                        string.Empty : Session["Session2"].ToString();
    }
```

（5）在浏览器中查看本页面，会出现如图 5.22 所示的界面。过 1 分钟后，单击“显示”按钮会出现如图 5.23 所示的界面。

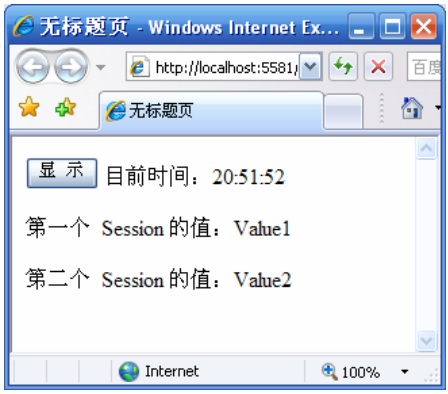


图 5.22　显示 Session 变量值图

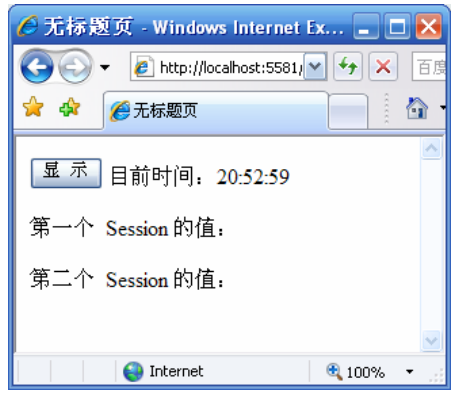


图 5.23　超时后 Session 变量值消失

注：第一次进入这个网页时，Session 对象变量的值会显示出来；接着不要做任何动作静待 1 分钟，1 分钟后单击“显示”按钮时，Session 对象变量的内容便被释放。

2. Session对象的事件

与 Application 对象一样，Session 对象也有 OnStart 和 OnEnd 事件。OnStart 事件在客户第一次从应用程序中请求 ASP.NET 网页时由 ASP.NET 调用，OnEnd 事件在会话关闭时调用。这些事件同样也只能在 Global.asax 文件中使用。

【例 5-5】 一个简单的在线人数计数器程序，利用 Applicaton 对象保存 Counter 变量，当有用户访问本站点，Counter 变量值就加 1。

设计步骤如下：

（1）运行 Visual Studio 2008，打开例 5-1 名为“InnerObject”的 ASP.NET 网站，并通过“添加新项”，在出现的窗口中选择“全局应用程序类”，名称无须修改。

（2）在 Global.asax 的 Application_Start 事件中，加入如下的代码：

```
Application.Lock();
Application["Counter"] = 0;                    //应用程序启动的时候，设置 Counter 变量值为 0
Application.UnLock();
```

（3）在 Global.asax 的 Sessoin_Start 事件中，加入如下的代码：

```
Application.Lock();
Application["Counter"] = (int)Application["Counter"] + 1;        //当新用户登录的时候，Counter 变量加 1
Application.UnLock();
```

（4）在网站内添加一个“Counter.aspx”的 Web 窗体，在其 Page_Load 方法内加入如下的代码：

```
Response.Write("你是第"+(int)Application["Counter"] + " 位访问者！ ");
```

（5）在浏览器中查看本页面，运行结果如图 5.24 所示的界面，再打开一个浏览器浏览本页面，会发现在线人数增加为 2 了。

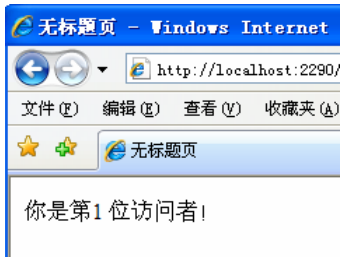


图 5.24　在线人数统计

L6. Server

Server 对象也是 Page 对象的成员之一，主要提供一些处理网页请求时所需的功能，如建立 COM 对象、将字符串编译码等。Server 对象的对象类名称是 HttpServerUtility。

Server 对象有以下两个属性：

① MachineName：服务器的计算机名称，为只读属性。

② ScriptTimeout：获取或设置程序执行的最长时间，即程序必须在该段时间内执行完毕，否则将自动终止，时间以秒为单位。系统的默认值为 90 秒。例如，ScriptTimeout = 100，表示最长程序执行时间为 100 秒。

表 5.17 列出了 Server 对象的常用方法。

表 5.17　Server 对象的常用方法

方　法	说　明
Transfer(url)	结束当前 ASP.NET 程序，然后执行参数 url 指定的程序
Execute(path)	执行由 path 指定的 ASP.NET 程序，执行完毕后仍继续原程序的执行
.CreateObject	创建服务器组件实例
HtmlDecode	将编码后的字符串译码回原来的 HTML 数据
HtmlEncode	将字符串编码为 HTML 可以辨识的信息
MapPath	传回实际路径和传入字符串的结合字符串，和 Request 对象的方法一样
UrlDecode	将编码后的 URL 的字符串译码
UrlEncode	将代表 URL 的字符串编码

1. HtmlEncode和HtmlDecode方法

当需要在网页上显示 HTML 标记时，若在程序语句中直接输出，则会被浏览器解译为 HTML 的内容，所以要通过 Server 对象的 HtmlEncode 方法将它编码再输出；而若要将编码后的结果译码回原本的内容，则使用 Server 对象的 HtmlDecode 方法。

【例 5-6】 HTML 编码练习，下列程序使用 Server 对象的 HtmlEncode 方法将“<B>HTML 内容</B>”编码后输出至浏览器，再利用 HtmlDecode 方法将把编码后的结果译码还原。

设计步骤如下：

（1）运行 Visual Studio 2008，打开例 5-1 命名为“InnerObject”的 ASP.NET 网站，新建命名为“HTMLEncodeAndDecode.aspx”的 Web 窗体文件。

（2）在 HTMLEncodeAndDecode.aspx 的 Page_Load 方法中添加如下的代码：

```
protected void Page_Load(object Sender, EventArgs e)
{
        string strHtmlContent;
        strHtmlContent = Server.HtmlEncode("<B>HTML 内容</B>");
        Response.Write(strHtmlContent);
        Response.Write("<P>");
        strHtmlContent = Server.HtmlDecode(strHtmlContent);
        Response.Write(strHtmlContent);
}
```

（3）在浏览器中查看本页，效果如图 5.25 所示。

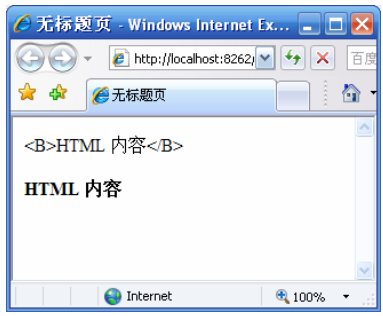


图 5.25　HTML 标记编码和解码

其实编码后的 HTML 标记变成了“<B>HTML 内容</B>”，这是因为<B>变成了“<B>”，</B>变成了“</B>”，所以才能在页面中显示出 HTML 标记。

2. UrlEncode和UrlDecode方法

在传递网页参数时是将数据附在网址后面传递，但是遇到一些如“#”、“&”的特殊字符，会读不到这些字符之后的参数。所以在需要传递特殊字符的场合，要先将欲传递的内容先以 UrlEncode 加以编码，才能够保证传递过去的值可以顺利被读到；而 UrlDecode 方法则是将编码过的内容译码还原。

【例 5-7】 Url 参数编码，下列程序使用两个 Html 对象来比较编码传递和未编码传递的结果，传递的参数内容是“a# @ #b”。

设计步骤如下：

（1）运行 Visual Studio 2008，打开例 5-1 命名为“InnerObject”的 ASP.NET 网站，新建命名为“ParameterEncode.aspx”和“ParameterDecode.aspx”的 Web 窗体文件。

（2）在 ParameterEncode.aspx 的源代码中加入如下的代码：

```
<form id="form1" runat="server">
<div>
    <a href="ParameterDecode.aspx?data1=a# @ #b">未编码的参数内容</a><br>
```

```
        <a href="ParameterDecode.aspx?data1=<%Response.Write(Server.UrlEncode("a# @ #b"));%>">编码过
            的参数内容</a>
    </div>
    </form>
```

（3）在 ParameterDecode.aspx 的 Page_Load 中加入如下的代码：

```
Response.Write(Request.QueryString["data1"]);
```

（4）在浏览器中查看 ParameterEncode.aspx 页面，运行后页面如图 5.26 所示，分别单击两个链接，会发现编码过的和未编码的传递的变量值是不同的。

若单击“未编码的参数内容”链接，传递过去的参数内容只显示 a，这和当初欲传递的参数内容并不相符。若单击“编码过的参数内容”链接，结果显示出正确的参数内容 a#@#b。可见对 URL 地址中的特殊字符需进行编码才可正确传递。

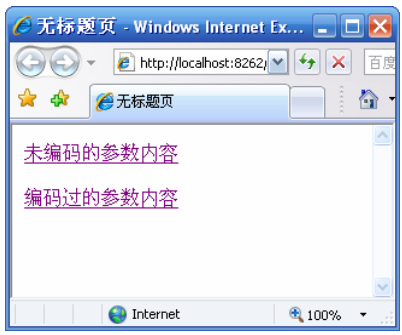


图 5.26　URL 地址编码和解码

L7. ViewState

ViewState 对象是 ASP.NET 中新增的一个对象，在以前的 ASP 中不存在此对象。此对象主要用来保存一个 ASP.NET 页面在用户请求的往返过程之间的页面信息与控件信息。

ViewState 对象是一个页面级别的对象，只能在当前页面内进行访问，不同用户访问同一个页面时会创建不同的 ViewState 对象。ViewState 对象的数据是保存在浏览器端的，当服务器在给客户呈现该页的 HTML 时候，会把回发过程中需要保留的页的当前状态和控件的值序列化为一个 Base64 编码的字符串，并以隐藏字段的方式输出到 HTML 代码中。当用户回发此页面的时候，隐藏字段会被重新发送给服务器，服务器会对其反序列化，再次呈现页面的时候，会使用反序列化的数据恢复当前页的状态和控件的值。

ViewState 对象中存储的常见数据类型有：字符串、整数、布尔值、Array 对象、ArrayList 对象、哈希表和泛型对象等。

使用 ViewState 可以带来很多方便，但是仍然要注意如下的问题：

① 视图状态信息将被序列化，然后使用 Base64 编码进行编码，这将生成大量的数据。在页回发到服务器时，视图状态信息将作为页回发信息的一部分发送。如果视图状态包含大量信息，则会影响页的性能。

② 虽然使用视图状态可以保存页和控件的值，但是在某些情况下，需要关闭视图状态。例如，使用 GridView 控件显示数据，单击 GridView 控件的“下一页”按钮，此时，

GridView 控件呈现的数据已经不再是前一页的数据，那么如果使用视图状态将前一页数据保存下来，不仅没有必要而且还会生成大量数据需要在回发过程中传递，增加了页面的体积。

③ 某些移动设备不允许使用隐藏字段。因此，视图状态对于这些设备无效。

【例 5-8】 利用 ViewState 保存控件和变量的值。

操作步骤如下：

（1）运行 Visual Studio 2008，打开例 5-1 命名为“InnerObject”的 ASP.NET 网站，新建命名为“ViewState.aspx”的 Web 窗体文件。

（2）在页面内添加一个 DropDownList 控件和两个 Button 控件，分别命名控件的名称为“DropDownListCity”、“ButtonAdd”和“ButtonSubmit”。

（3）双击“ButtonAdd”，进入 Click 事件代码中，加入如下的代码：

```
ListItem item1 = new ListItem();
item1.Text = "南京";
item1.Value = "南京";
ListItem item2 = new ListItem();
item2.Text = "上海";
item2.Value = "上海";
DropDownListCity.Items.Add(item1);
DropDownListCity.Items.Add(item2);
```

（4）在浏览器中查看本页，页面第一次请求的时候，会发现 DropDownListCity 内部没有项；单击“ButtonAdd”，DropDownListCity 内出现我们添加的项；再次单击“ButtonSubmit”，会发现 DropDownListCity 内部还有项目。

（5）如果把 DropDownListCity 控件的 EnableViewState 属性设置为 False，再次运行第（4）步，当再次单击“ButtonSubmit”，会发现“DropDownListCity”内部项目丢失。

（6）双击“ButtonAdd”，在 Click 事件代码中添加如下的代码：

```
ViewState["Add"] = true;
```

（7）双击“ButtonSubmit”，在 Click 事件代码中添加如下的代码：

```
if (ViewState["Add"] != null)
    Response.Write(ViewState["Add"].ToString());
```

（8）在浏览器中查看本页，单击“ButtonAdd”，再单击“ButtonSubmit”，会发现页面上输出“True”。

结论：ASP.NET 会为 ASP.NET 页面或页面内的控件启用 EnableViewState 属性，用来保存页面往返过程中页面内视图的状态。

可以禁用整个页面的 ViewState，也可以禁用某一个控件的 ViewState。禁用页面的 ViewState，可以在页面的 Page_load 内加入如下的代码：

```
Page.EnableViewState = false;
```

也可以在页面源视图下，给@Page 指令加上 EnableViewState="false"属性。

第 6 章　系统基础数据维护模块

◇ 任务目标

本章主要目标是完成系统基础数据的维护工作。本系统中存在教师信息、课程信息、班级信息、学生信息、课程安排信息等基础数据，维护方式基本相同。这里通过实现其中的某个或某几个来说明基础数据维护的方式。完成后的数据维护界面如图 6.1、图 6.2、图 6.3 所示。

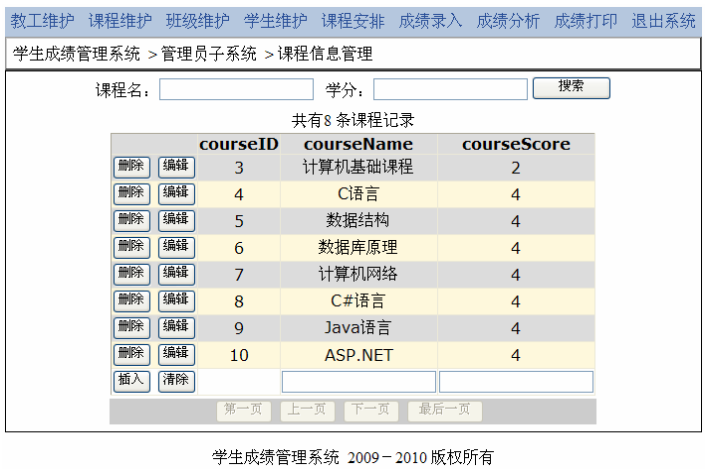


图 6.1　课程信息维护

图 6.2　教师信息维护

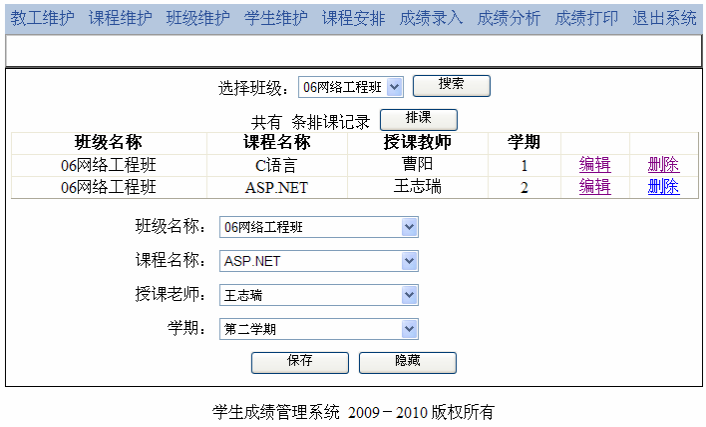


图 6.3　排课信息维护

第一部分　应 用 实 践

6.1　完成课程信息的维护

基础数据的维护工作主要分为查询、新增、修改、删除等几个基本功能。由于功能比较简单，因此这里把所有的功能都在一个页面内实现。

ASP.NET中提供了很多种访问数据库的方式，每种方式都有各自的特点，这里就以几个数据表的维护为例分别讲解常用的几种实现方法。

采用 **SqlDataSource**+ **ListView** 的方式来完成课程信息的维护，用 SqlDataSource 实现与数据库之间的访问，用 ListView 提供所需的数据查询、修改、插入、删除视图。

1．规划课程信息维护界面

课程信息维护页面的基本布局如图 6.4 所示。

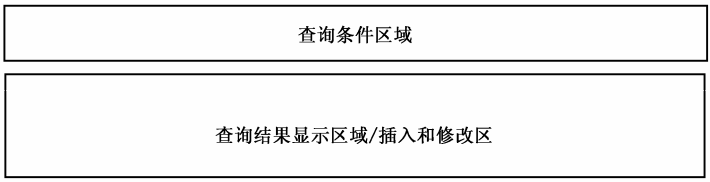


图 6.4　课程信息维护页面的基本布局

2．设计数据维护界面

（1）打开前面创建的网站项目，右击“解决方案资源管理器”内的 Admin 文件夹，选择“添加新项”，在弹出的窗口中选择“Web 窗体”，命名为“course.aspx”，语言选择“Visual C#”，选中“选择母版页”复选框，完成后单击“添加”按钮，在出现的窗口中选择 Admin 文件夹下的 adminMasterPage.master，单击“确定”按钮完成新页面的创建。

（2）打开 Web.sitemap 文件，在<siteMapNode url="" title="管理员子系统" description="">标记内部加入如下的内容：

```
<siteMapNode url="~/admin/course.aspx" title="课程信息管理"    description="" />。
```

（3）切换到 course.aspx 页面的设计视图，打开属性窗口，在窗口的下拉框中选择“Document”，设置 StyleSheetTheme 属性为“主题 1”。

（4）切换到“源”视图，把光标定位到ID 为“Content2”的标记内，双击“工具栏”中 HTML 内的“DIV”，并设置其 ID 为“content”，设置 Style 属性值为“border-style: solid; border-width: 1pt”。

（5）打开页面的“源”视图，把光标定位到ID 为“content”层内部，双击“工具栏”中 HTML 内的“DIV”，在页面上插入一个层，设置其 ID 属性为“search”。切换到“设计”视图，把光标定位到此层内，单击“应用样式”窗口中的“innerLayout”样式规则，给该层应用样式规则。

（6）把光标定位到 ID 为“search”的 DIV 内，如图 6.5 所示向 DIV 内添加控件。

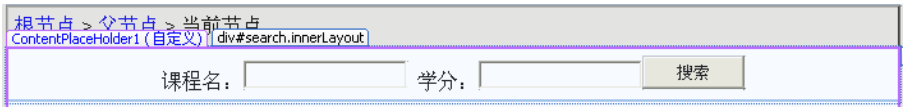


图 6.5　搜索栏设计

从左到右分别设置 DIV 内控件的属性，如表 6.1 所示。

表 6.1　控件属性设置

控 件 类 型	相关属性和值
Textbox	ID=“ searchTextBoxcourseName”
Textbox	ID=“ searchTextBoxcourseScore”
Button	ID=“ searchButton” Width= “80 px”

（7）在“源”视图中，把光标定位到 ID 为“search”的 DIV 标记的下部，双击“工具栏”中 HTML 中的 DIV，在光标处插入一个 DIV。选中此 DIV，设置其 ID 为“result”，在“应用样式”窗口中，单击“innerLayout”样式规则，给该层应用样式规则。

（8）把光标定位到 ID 为“result”的 DIV 内，如图 6.6 所示向 DIV 内添加控件。

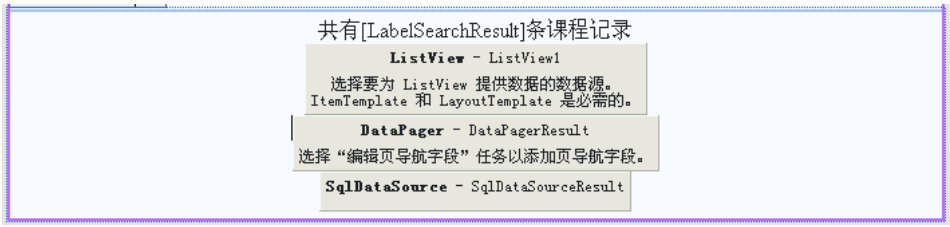


图 6.6　搜索结果栏设计

从左到右分别设置 DIV 内控件的属性，如表 6.2 所示。

表 6.2　控件属性设置

控 件 类 型	相关属性和值
Label	ID=“ LabelSearchResult”,Text=“”
ListView	ID=“ListViewResult”
DataPager	ID=“ DataPagerResult”
SqlDataSource	ID=“SqlDataSourceResult”

（9）选中“SqlDataSourceResult”控件，单击右上角“SqlDataSource 任务”中的“配置数据源”选项，在弹出的“配置数据源”对话框的下拉框中选择“SqlServer2005ConnectionString”，单击“下一步”按钮，进入“配置 Select 语句”对话框，在“指定来自表或视图的列”的下拉框选中“Course”表，在列中，选中 courseID、courseName 和 courseScore，则所生成的 Select 语句为“SELECT [courseID], [courseName], [courseScore] FROM [Course]”，单击“下一步”按钮，进入“测试查询”窗口。单击“测试查询”按钮，查看查询出的结果是否满足要求，如果满足，则单击“完成”按钮；如果不满足，则返回上一个窗口进行修改。

注：在“配置 Select 语句”对话框中，如果是单表查询，则通过“指定来自表或视图的列”的下拉框选中“对应的表”，并在“列”多选框内选择要查询出的列。如果是多表查询，则通过“指定自定义SQL 语句和存储过程”自己编写要查询的 SQL 语句。

（10）选中“SqlDataSourceResult”控件，单击右上角“SqlDataSource 任务”中的“配置数据源”选项，在弹出的“选择您的数据连接”对话框中单击“下一步”按钮，进入“配置 Select 语句”对话框，选择“指定自定义 SQL 语句和存储过程”，单击“下一步”按钮，在弹出的“定义自定义语句和存储过程”对话框在中分别选择“UPDATE”、“INSERT”、“DELETE”选项卡，在对应的文本框中输入如下 SQL 的语句：

```
UPDATE Course SET courseName = @courseName,courseScore = @courseScore WHERE courseID =
        @courseID
INSERT INTO Course(courseName,courseScore) VALUES(@courseName,@courseScore)
DELETE FROM Course WHERE courseID=@courseID
```

单击“下一步”、“完成”按钮。

（11）选中“ListView”控件，单击右上角“ListView 任务”，在“选择数据源”下拉框中选择“SqlDataSourceResult”控件。单击“配置 ListView”按钮，在弹出的“配置 ListView”对话框中选择“网格”布局，选择“专业型”样式，并选中“启用编辑”、“启动插入”、“启动删除”和“启动分页”选项，单击“确定”按钮完成 ListView 配置。

（12）打开“ListViewResult”的属性窗口，设置其 PagedControlID 为“ListView-Result”，PageSize 为“10”，设置后的界面效果如图 6.7 所示。

3．编写本页面所需的相关代码

（1）单击搜索栏中的按钮后，会根据用户录入的信息进行查询，并把查询的结果显示在下部的 GridView 中，因此“搜索”按钮的 Click 事件需要添加代码。

图 6.7　页面外观

双击“搜索”按钮，在对应的事件中添加如下的代码：

```
//首先定义需要查询的内容，具体查询的条件根据用户的录入自动生成
string sql = "SELECT courseID, courseName, courseScore FROM Course where 1=1 ";
//如果录入课程名称，则根据课程名称进行模糊查找
if (this.searchTextBoxcourseName.Text != "")
{
    sql += " and courseName LIKE '%"+ this.searchTextBoxcourseName.Text.Trim() +"%'";
}
//如果录入课程学分，则根据课程学分查找
if (this.searchTextBoxcourseScore.Text != "")
{
    sql += " and    courseScore = '"+ this.searchTextBoxcourseScore.Text.Trim() + "'";
}
this.SqlDataSourceResult.SelectCommand = sql;
```

（2）页面初次加载的时候，GridView 中只显示数据库的前 6 条记录，查看更多的数据，需要通过“查询”按钮完成，因此需要在 Page_load 事件中添加如下的代码：

```
if (Session["userType"].ToString() != "0")   //如果不是管理员则返回
{   Response.Write("<script language=javascript>history.go(-1);</script>");}      //重定向到前面页面
//页面第一次请求的时候执行
if (!Page.IsPostBack)
  {
      string sql = "SELECT   TOP 6   courseID,courseName,courseScore
                                  FROM Course ORDER BY courseID DESC";
      this.SqlDataSourceResult.SelectCommand = sql;
  }
```

（3）为了能够在 Label 中显示查询出的记录条数，需要在 SqlDataSourceResult 控件的 Selected 事件中添加代码：选中“SqlDataSourceResult”控件，右击此控件并选择“属性”选项，在打开的属性窗口中单击顶部 图标，在列表中双击“Selected”，进入其“Selected 事件”，并在内部添加如下的代码：

```
//当 SqlDataSource 执行完查询操作后所触发的一个事件
protected void SqlDataSourceResult_Selected(object sender, SqlDataSourceStatusEventArgs e)
{
    //参数 e 的 AffectedRows 属性表示查询出的记录数目
    this.LabelSearchResult.Text = e.AffectedRows.ToString();
}
```

4．在浏览器中浏览本页

（1）切换到“设计”视图，右击页面并选择“在浏览器中查看”，页面如图 6.8 所示。

图 6.8　课程维护界面

（2）在 courseName 和 courseScore 下部的文本框中输入相应的课程名称和课程学分，单击“插入”按钮，则一条记录就插入到数据库中，连续插入几个课程后，界面如图 6.9 所示。

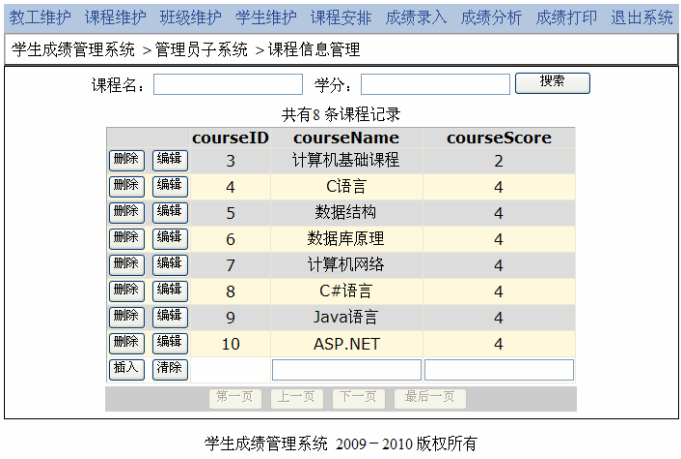


图 6.9　课程维护界面

（3）单击某条记录的“编辑”按钮，则该行将变成可编辑状态，修改结束后，单击“更新”按钮可以把修改后的内容插入到数据表中，也可以单击“取消”按钮取消本次的修

改操作，如图 6.10 所示。单击某条记录左部的“删除”按钮，则可以把本条记录从数据表内删除。

图 6.10　修改视图

（4）在搜索栏中“课程名”处输入课程名称的部分文字，单击“搜索”按钮就可以查询出对应的课程，也可以在“学分”处输入一个学分，则将查询出本学分所有的课程；也可以同时通过“课程名”和“学分”进行查询。

6.2　完成教师信息的维护

采用 SqlDataSource+ListView 的方式比较适合字段比较少的表进行数据维护，当字段很多的时候，编辑界面会显示很长，因此需要采用专门用于编辑的控件 **FormView 控件**。

这里采用 **GridView**+SqlDataSource+FormView 的方式完成教师信息的维护，其中 GridView 用于表格方式显示出多条记录，SqlDataSource 用于实现与数据库之间的访问，FormView 用于实现单条记录的查看、修改、插入等操作。

1．规划教师信息维护界面

教师信息维护页面的基本布局如图 6.11 所示。

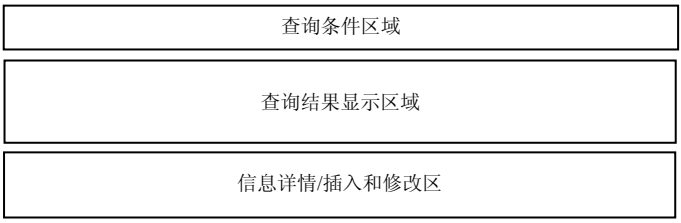


图 6.11　页面基本布局

2．设计数据维护界面

（1）在前面所创建的网站项目中打开 teacher.aspx 页面的的“源”视图，删除 ID 为“Content2”的标记内的前面所输入的“内容页内容”。把光标定位到 ID 为“Content2”的标记内，双击“工具栏”中 HTML 内的“DIV”，在光标处插入一个层，并设置其 ID 为“content”，设置 Style 属性值为“border-style: solid; border-width: 1pt”。

（2）把光标定位到ID 为“content”的层内，双击“工具栏”中 HTML 内的“DIV”，在页面上插入一个层，设置其 ID 属性为“search”。把光标定位到此层内，单击“应用样式”窗口中的“innerLayout”样式规则，为该层应用样式规则。

（3）把光标定位到 ID 为“search”的 DIV 内，按照如图 6.12 所示向 DIV 内添加控件。

从左到右分别设置 DIV 内控件的属性，设置如表 6.3 所示。

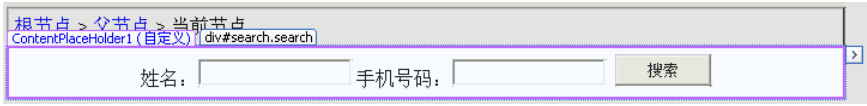


图 6.12　搜索栏设计

表 6.3　控件属性设置

控 件 类 型	相关属性和值
Textbox	ID=“searchTextBoxteaName”
Textbox	ID=“searchTextBoxteaTelphone”
Button	ID=“searchButton” Width= “80 px”

在实际应用中可以根据需要、用于查询的记录数目，修改查询界面的外观。

（4）按照上面的添加 DIV 方法在 ID 为“search”的 DIV 标记的下部添加一个 DIV，其 ID 为“result”，在“应用样式”窗口中，单击“innerLayout”样式规则，为该层应用样式规则。

（5）把光标定位到 ID 为“result”的 DIV 内，按照如图 6.13 所示向 DIV 内添加控件。

图 6.13　搜索结果栏设计

从左到右分别设置 DIV 内控件的属性，设置如表 6.4 所示。

表 6.4　控件属性设置

控 件 类 型	相关属性和值
Label	ID=“LabelSearchResult”
Button	ID=“AddButton”，Width=“80 px”
GridView	ID=“GridViewResult” Width=“100%” AllowPaging=“True”
SqlDataSource	ID=“SqlDataSourceResult”

（6）打开“SqlDataSourceResult”控件属性窗口，单击“ConnectionString”属性右边的下拉框，设置其值为“SqlServer2005ConnectionString”。单击 SelectQuery 右边的按钮，打开“命令和参数编辑器”窗口，在窗口内输入如下的 Sql 语句：

```
SELECT    top (5)    teaID, teaName, teaTelphone, teaPhone,teaEmail,teaType FROM Teacher
```

单击 DeleteQuery 右边的按钮，打开“命令和参数编辑器”窗口，在窗口内输入如下的 Sql 语句：

```
DELETE FROM Teacher WHERE teaID=@ teaID
```

（7）选中 GridViewResult，单击右上角“GridView 任务”中的“选择数据源”下拉框，选择“SqlDataSourceResult”控件。单击“刷新架构”选项重新刷新架构，单击“GridView 任务”中的“编辑列”选项，打开**“字段”对话框**，在“选中的字段”列表中选中“teaID”，在右栏中设置其 HeaderText 属性为“编号”，分别设置其他字段的 HeaderText 为“姓名”、“电话”、“手机”、“邮箱”和“类型”。

（8）在“可用字段”窗口中选中“TemplateField”字段，并单击“添加”按钮把其添加到“选定的字段”中，并设置其 HeaderText 为“操作”。在“选定的字段”列表中选中“类型”字段，单击右边的“将此字段转换为 TemplateField”按钮，单击“确定”按钮关闭对话框。

（9）右击 GridView 控件，选择“编辑字段”内的“Column5-类型”，选中“ItemTemplate”中的 Label 控件，单击右上角的图标，在打开的“Label 任务”中单击“编辑 DataBindings”选项，弹出“Label1 DataBinding”对话框，选中左侧的“Text”，单击右侧的“自定义绑定”，把“**Bind**("teaType")”改为“teaType(**Eval**("teaType"))”，单击“确定”按钮，关闭对话框。

（10）右击 GridView 控件，选择“编辑字段”内的“Column6-操作”，“ItemTemplate”中添加 3 个 LinkButton 控件，按表 6.5 设置属性，设置后的效果如图 6.14 所示。

表 6.5　控件属性设置

控 件 类 型	相关属性和值
LinkButton1	ID=“LinkButton1”，Text=“详情”，CommandName=“MyDetail”
LinkButton2	ID=“LinkButton2”，Text=“编辑”，CommandName=“MyModify”
LinkButton3	ID=“ LinkButton3”，Text=“删除”，CommandName=“MyDelete”

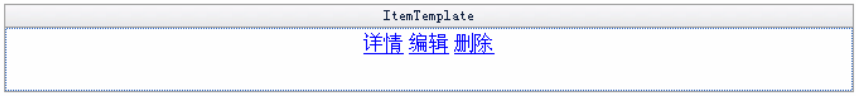


图 6.14　操作视图设计

（11）切换到“源”视图，在 ID 为“result”的 DIV 标记的下部添加一个 DIV，设置其 ID 为“Operation”，在“应用样式”窗口中，单击“innerLayout”样式规则，为该层应用样式规则。

（12）把光标定位到 ID 为“Operation”的 DIV 内，按照如图 6.15 所示向 DIV 内添加控件。

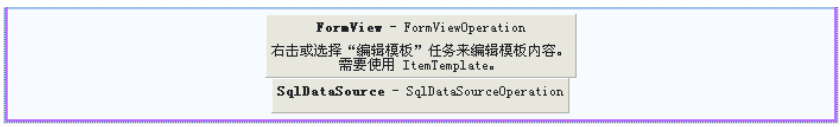


图 6.15　操作栏设计

从左到右分别设置 DIV 内控件的属性，设置如下表 6.6 所示：

表 6.6　控件属性设置

控 件 类 型	相关属性和值
FormView	ID="FormViewOperation",Width="100%"
SqlDataSource	ID="SqlDataSourceOperation"

（13）切换到“设计”视图，打开“SqlDataSourceOperation”的属性窗口，设置其 ConnectionStrings 为“SqlServer2005ConnectionString”，设置其 SelectQuery 为：

```
SELECT teaID,teaName,teaSex,teaTelphone,teaType,teaPhone,teaEmail,teaPassword FROM Teacher
```

设置 InsertQuery 为：

```
INSERT INTO Teacher(teaName,teaSex,teaTelphone,teaType,teaPhone,teaEmail,teaPassword,)
VALUES(@teaName,@teaSex,@teaTelphone,@teaType,@teaPhone,@teaEmail,@teaPassword)
```

设置 UpdateQuery 为：

```
UPDATE Teacher SET teaName=@teaName,teaSex=@teaSex,teaTelphone=@teaTelphone,
    teaType=@teaType,teaPhone=@teaPhone,teaEmail=@teaEmail,teaPassword=@teaPassword where
    teaID=@teaID
```

（14）选中 FormViewOperation，在“FromView 任务”中，设置数据源为“SqlDataSourceOperation”，单击“刷新架构”按钮，显示如图 6.16 所示的界面。

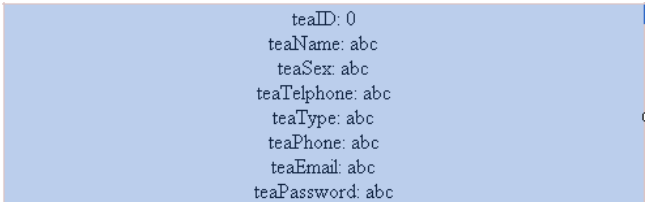


图 6.16　ItemTemplate 模板

（15）右击选择“FormViewOperation”→“编辑模板”→“ItemTemplate”，切换到“**Item-Template**”模板，在模板顶部插入一个 DIV，通过“应用样式”设置其样式为“innerLayout”，把光标定位到本层内，选择“表”→“插入表”菜单项，弹出“插入表格”对话框，设置行数为“4”，列数为“4”，其他采用默认值，插入表后，将 ItemTemplate 模板内的内容按照如图 6.17 所示摆放。

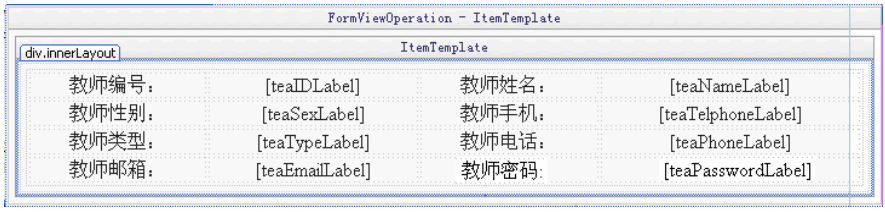


图 6.17　ItemTemplate 模板设计效果

（16）在 ItemTemplate 模板中选择 teaSexLabel 的 Label 任务中的“编辑 DataBinding”

选项，通过自定义绑定，设置 Text 属性绑定为“teaSex(Eval("teaSex"))”。采用同样的方式，设置 teaTypeLabel 的 Text 属性绑定为“teaType(Eval("teaType"))”。

（17）在 ItemTemplate 模板内分别选中表格内的第一列或第三列，在“应用样式”窗口中选中“.right”样式。按照同样的方法分别选中表格内的第二列或第四列，在“应用样式”窗口中选中“.left”样式，应用样式后的外观如图 6.18 所示，右击“FormViewOperation”，选择“结束模板编辑”，完成 ItemTemplate 模板制作。

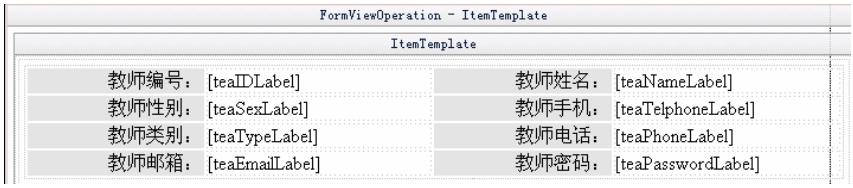


图 6.18　ItemTemplate 模板样式效果

（18）以同样的方式编辑 FormViewOperation 的 InsertItemTemplate 模板和 EditItemTemplate 模板，插入层，插入表格，重排元素的位置，设置后的外观如图 6.19、图 6.20 所示。

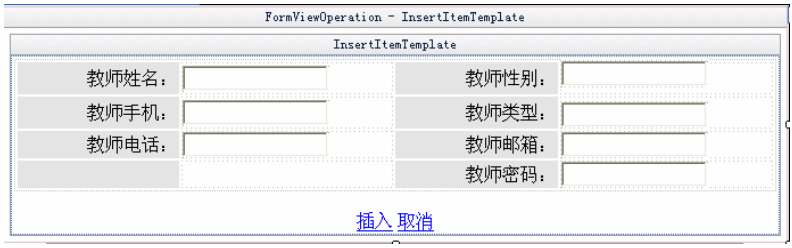


图 6.19　InsertItemTemplate 模板样式效果

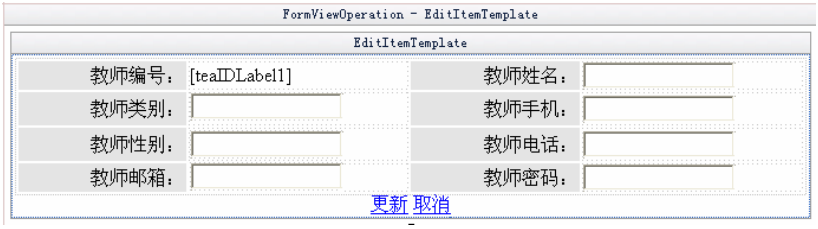


图 6.20　EditItemTemplate 模板样式效果

（19）打开 FormViewOperation 的 InsertItemTemplate 模板，删除教师性别右侧的文本框，插入一个“**RadioButtonList**”控件，通过 Items 属性插入两项，设置第一项的 Text 属性为“男”，Value 属性为“0”；设置第二项的 Text 属性为“女”，Value 属性为“1”，设置其 RepeatDirection 为“Horizontal”。在“RadioButtonList 任务”中选择“编辑 DataBindings”选项，在弹出的对话框中设置 SelectedValue 属性绑定到“teaSex”字段，格式为“{0:G}”，按照同样的方式修改 EditItemTemplate 模板内的教师性别。

（20）打开 FormViewOperation 的 InsertItemTemplate 模板，删除教师性别右侧的文本框，插入一个“**RadioButtonList**”控件，通过 Items 属性插入两项，设置第一项的 Text 属性为“管理员”，Value 属性为“0”；设置第二项的 Text 属性为“教师”，Value 属性为“1”，设置

其 RepeatDirection 为“Horizontal”。在“RadioButtonList 任务”中选择“编辑 DataBindings”，在弹出的对话框中设置 SelectedValue 属性绑定到“teaType”字段，按照同样的方式修改 EditItemTemplate 模板内的教师类别。设置后的 InsertItemTemplate 模板效果如图 6.21 所示。

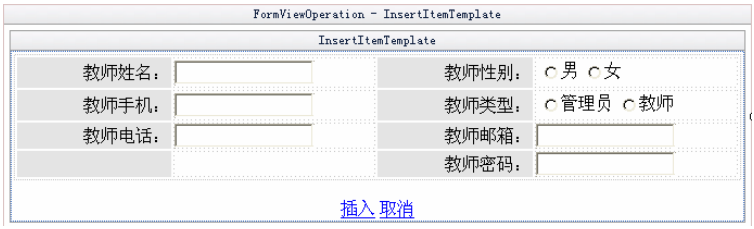


图 6.21　InsertItemTemplate 模板最终效果

3．编写本页面所需的相关代码

（1）默认情况下，GridView 内只显示5 条教师信息，单击“搜索”按钮后，应该根据用户的录入情况进行搜索，把搜索出的所有信息显示到 GridView 内。双击“搜索”按钮，添加 Click 事件代码，代码如下所示：

```
//初始把所有的都查询出来
string sql = " SELECT teaID, teaName, teaTelphone, teaPhone, teaEmail, teaType
                FROM Teacher where 1=1 ";
//如果姓名处有输入，则根据姓名字段进行过滤
if (this.searchTextBoxteaName.Text != "")
{
    sql = sql + " and teaName like '%" + this.searchTextBoxteaName.Text.Trim() + "%' ";
}
//如果手机处有输入，则根据手机字段进行过滤
if (this.searchTextBoxteaTelphone.Text != "")
{
    sql = sql + " and teaTelphone like '%" + this.searchTextBoxteaTelphone.Text.Trim() + "%' ";
}
this.SqlDataSourceResult.SelectCommand = sql;
this.GridViewResult.DataSourceID = this.SqlDataSourceResult.ID;
//把对应的查询语句保存到 ViewState 对象中，当页面回送的时候使用
ViewState["sql"] = sql;
```

（2）GridView 控件内模板字段内的控件不包含所在行的索引值，当其触发 RowCommand 事件时无法获取由哪行触发的，因此需要通过程序代码在该行创建的时候为其设置 CommandArgument 值，以及在 RowCreated 事件中为所需要的行添加，这里需要为上文所添加的 LinkButton1、LinkButton2、LinkButton3 添加 CommandArgument 值，右击“GridViewResult”，选择“属性”，双击“事件”内的 RowCreated 事件，进入其编写代码位置，在其中加入如下的代码：

```
    //如果行的类型是数据行，需要为此行内模板列内的控件设置 CommandArgument 值
    //此值可能用来判断用户单击的是哪行内的控件
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        //在当前行内获取到控件的引用，通过行的索引设置控件的 CommandArgument
        LinkButton LinkButton1 = (LinkButton)e.Row.FindControl("LinkButton1");
        LinkButton1.CommandArgument = e.Row.RowIndex.ToString();
        LinkButton LinkButton2 = (LinkButton)e.Row.FindControl("LinkButton2");
        LinkButton2.CommandArgument = e.Row.RowIndex.ToString();
        LinkButton LinkButton3 = (LinkButton)e.Row.FindControl("LinkButton3");
        LinkButton3.CommandArgument = e.Row.RowIndex.ToString();
    }
```

（3）当用户单击“新增教工”按钮时，FormViewOperation 应该显示对应的“插入模板”，录入结束后，单击“插入”按钮，实现把数据插入到数据库中。双击“新增教工”按钮，此按钮的 Click 事件代码如下所示：

```
    //更改 FormView 的显示视图
    this.FormViewOperation.ChangeMode(FormViewMode.Insert);
```

（4）由于数据库中对于教工类型和性别的描述都通过编码表示的，但是显示给客户的时候不能直接显示，需要根据数据库的编码转化为方便理解的值，因此需要编写如下两个函数，函数相关代码如下：

```
    //根据数据库内的编码返回所代表的含义
    public string teaType(object source)
    {
        if (source.ToString() == "0")
            return "管理员";
        if (source.ToString() == "1")
            return "教工";
        return "";
    }
    //根据数据库内的编码返回所代表的含义
    public string teaSex(object source)
    {
        if (source.ToString() == "0")
            return "男";
        if (source.ToString() == "1")
            return "女";
        return "";
    }
```

（5）Page_Load 方法中添加代码，方便查询结束后，当页面再次回送的时候，GridView 中还能够显示查询过的结果。

```
if(Session["userType"].ToString() != "0")                                    //如果不是管理员则返回
{     Response.Write("<script language=javascript>history.go(-1);</script>");    }     //重定向到前面页面
if (ViewState["sql"] != null)
{
        string sql = ViewState["sql"].ToString();
        this.SqlDataSourceResult.SelectCommand = sql;
        this.GridViewResult.DataSourceID = this.SqlDataSourceResult.ID;
}
```

（6）为每行的“详情”按钮和“编辑”按钮编写事件，当用户单击“详情”按钮时，FormViewOperation 内显示对应行的详情视图；当单击“编辑”按钮时，FormViewOperation 内显示对应行的编辑视图。在 GridView 中单击按钮都是触发 RowCommand 事件，在事件内部通过命令名称判断是哪个按钮触发的，不同的按钮对应不同的代码。打开 GridView 控件的属性窗口，单击 图标，双击“RowCommand”，在此方法中添加如下代码：

```
//如果单击的是详情按钮
if (e.CommandName == "MyDetail")
{
    //获取单击行的索引值
    int index = Convert.ToInt32(e.CommandArgument);
    //通过行索引获取此行的教工编号
    string teaID = this.GridViewResult.Rows[index].Cells[0].Text.Trim();
    //创建 Sql 语句，从数据库内查询出此教工
    string sql = "SELECT teaID,teaName,teaSex,teaTelphone,teaType,teaPhone,teaEmail,teaPassword
                        FROM Teacher WHERE teaID='" + teaID + "'";
    //设置 SqlDataSource 数据源控件，使用此 Sql 语句进行查询
    this.SqlDataSourceOperation.SelectCommand = sql;
    //设置 FromView 控件的数据源控件
    this.FormViewOperation.DataSourceID = this.SqlDataSourceOperation.ID;
}
if (e.CommandName == "MyModify")
{
    int index = Convert.ToInt32(e.CommandArgument);
    string teaID = this.GridViewResult.Rows[index].Cells[0].Text.Trim();
    string SelectSql = "SELECT teaID,teaName,teaSex,teaTelphone,teaType,teaPhone,teaEmail,
          teaPassword FROM Teacher WHERE teaID='" + teaID + "'";
    this.SqlDataSourceOperation.SelectCommand = SelectSql;
    this.FormViewOperation.DataSourceID = this.SqlDataSourceOperation.ID;
    //更改 FormView 的显示视图
    this.FormViewOperation.ChangeMode(FormViewMode.Edit);
}
```

（7）为了方便查看搜索后查询出几条记录，需要给 SqlDataSourceResult 控件的 Selected 事件添加代码。打开 SqlDataSourceResult 控件的属性窗口，单击 图标，双击“Selected”，在此方法中添加如下代码：

```
this.LabelSearchResult.Text = e.AffectedRows.ToString();
```

4．在浏览器中使用本页

（1）在“设计视图”中，右击选择“在浏览器中查看”，出现如图 6.22 所示的界面。

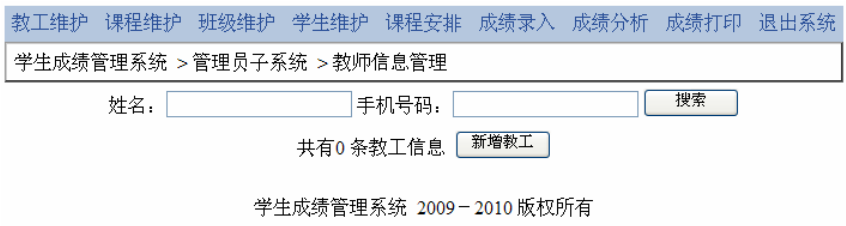


图 6.22　初始界面

（2）单击“新增教工”按钮，出现“新增视图”，如图 6.23 所示，在界面内录入数据后，单击“插入”按钮，则向数据库表内插入了一条记录，重复此操作可以向数据表内插入多条记录，单击“搜索”按钮，显示刚才所插入的记录，界面如图 6.24 所示。

图 6.23　新增教工信息界面

图 6.24　增加教工后的界面

（3）单击每条记录右边的“详情”按钮，下部就会显示出此记录的详细信息。单击

“编辑”按钮，则显示“编辑视图”，如图 6.25 所示，对信息进行修改后，单击“更新”按钮实现数据的更新操作。若单击“删除”按钮，则此条记录将会从数据表内删除，实现删除操作。

图 6.25　编辑教工信息界面

（4）当表内记录很多的时候需要利用搜索功能显示出需要查看的记录，在搜索栏“姓名”处输入的所要查询的教工姓名（支持模糊匹配），或在“手机号码”处输入所要查询的教工手机（支持模糊匹配），单击“搜索”按钮，下方将显示出所有匹配的记录，如图 6.26 所示。

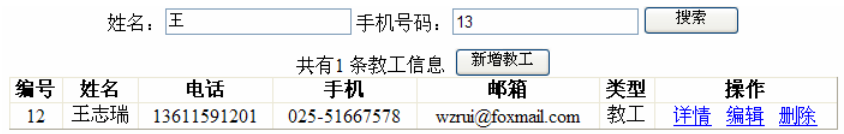


图 6.26　搜索教工界面

6.3　完成课程安排的维护

采用 SqlDataSource+GridView+FormView 的方式，比 SqlDataSource+ListView 方式功能强大一些，方便处理信息比较多的情况，但是由于数据是通过 SqlDataSource 实现与数据库的交互，因此在数据的处理方面不是很灵活，只是通过简单的单条 SQL 指令实现数据库的操作。如果用户输入的数据需要进行转换后再插入数据库，或者一次操作要进行多条的数据操作，就不是很方便实现。因此对于复杂的数据处理问题应该采用“控件和编码”的方式实现。

这是一种灵活的模式，但也是工作量最多的一种方式，查询的结果页面采用 GridView 控件展现，记录的详情、新增、修改界面通过 ASP.NET 提供的 Web 服务器控件设计完成，用户新增或修改的时候把控件内的内容通过 ADO.NET 数据访问方法将数据插入到数据库中，这种模式所有的工作都通过代码独自完成，因此相对来说比较繁琐，但是功能实现起来比较灵活。

1．规划课程安排维护的界面

课程安排维护页面的基本布局如图 6.27 所示。

2．设计课程安排维护的页面

（1）打开前文所创建的网站项目，在 Admin 文件夹中创建一个命名为“Arrange-Course.aspx”页面，并套用 Admin 文件夹内的母版页 adminMasterPage.master。

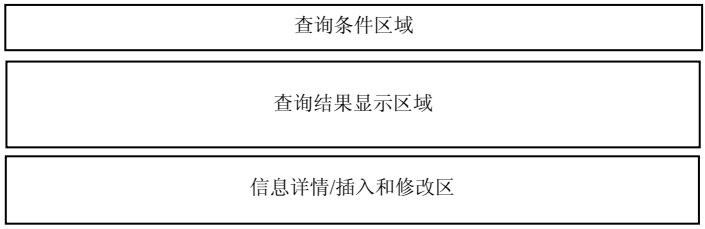


图 6.27　页面基本布局

（2）打开 Web.sitemap 文件，在<siteMapNode url="" title="管理员子系统" description="" >标记内部加入如下的内容：<siteMapNode url="~/admin/ArrangeCourse.aspx" title="课程安排维护" description="" />。

（3）切换到 ArrangeCourse.aspx 页面的设计视图，在“属性”窗口中的下拉框中选择“Document”，设置 StyleSheetTheme 属性为“主题 1”。

（4）切换到“源”视图，把光标定位到 ID 为“Content2”的标记内，双击“工具栏”中 HTML 内的“DIV”，在光标处插入一个层，并设置其 ID 为“content”，设置 Style 属性值为“border-style: solid; border-width: 1pt”。

（5）按照上面的方法在ID 为“content”的层内添加一个 DIV，其 ID 属性为“search”。切换到“设计”视图，把光标定位到此层内，单击“应用样式”窗口中的“innerLayout”样式规则，为该层应用样式规则。

（6）按照如图 6.28 所示向 ID 为“search”的 DIV 内添加控件。

图 6.28　搜索栏设计

从左到右分别设置 DIV 内控件的属性，设置如表 6.7 所示。

表 6.7　控件属性设置

控 件 类 型	相关属性和值
DropDownList	ID=“DropDownListClass”
Button	ID=“ButtonSearch”,width=“80 px”，Text=“搜索”
SqlDataSource	ID=“SqlDataSourceClass”

（7）通过 SqlDataSource 任务，打开“配置数据源”对话框，在“应用程序连接数据库应该使用哪个连接”处，选择“SqlServer2005ConnectionString”，单击“下一步”按钮，在弹出的“配置 Select 语句”对话框中指定来自表“Class”，选中“classID”和“className”两个列，单击“下一步”、“完成”按钮，完成数据源的配置。通过 DropDownList 任务，选择数据源为“SqlDataSourceClass”，“显示的字段”为“className”，“值字段”为“classID”，单击“确定”按钮，完成搜索栏的设置。

（8）切换到“源”视图中，在 ID 为“search”的 DIV 标记的下部添加一个 DIV。设置其 ID 为“result”，在“应用样式”窗口中，单击“innerLayout”样式规则，为该层应用样式规则。

（9）把光标定位到 ID 为“result”的 DIV 内，按照如图 6.29 所示向 DIV 内添加控件。

图 6.29　搜索结果栏设计

从左到右分别设置 DIV 内控件的属性，设置如表 6.8 所示。

表 6.8　控件属性设置

控 件 类 型	相关属性和值
Label	ID=“LabelSearchResult”
Button	ID=“ButtonAdd”，Width=“80 px”
GridView	ID=“GridView1”，Width=“100%”，AllowPaging=“True”
SqlDataSource	ID=“SqlDataSourceResult”

（10）通过 SqlDataSource 任务为 SqlDataSourceResult 控件设置数据库连接串为“SqlServer-2005ConnectionString”，Select 语句为“SELECT Class.className, Course.course-Name, Teacher.teaName, Arrange.team, Arrange.classID, Arrange.courseID, Arrange.teaID FROM Arrange INNER JOIN Class ON Arrange.classID = Class.classID INNER JOIN Course ON Arrange.courseID = Course.courseID INNER JOIN Teacher ON Arrange.teaID = Teacher.teaID WHERE (Arrange.classID = @classID)”，@classID 的参数源为“Control”，ControlID 为“DropDownListClass”。

（11）通过 GridView 任务为 GridViewResult 配置数据源为“SqlDataSourceResult”控件，通过“编辑列”，打开“字段”窗口，删除 classID、courseID 和 teaID 字段，分别设置其他的字段的名称为“班级名称”、“课程名称”、“授课老师”和“学期”，添加 2 个 ButtonField 到“选定的字段中”，分别设置 ButtonField 的 Text 属性为“编辑”和“删除”，ButtonField 的 CommandName 属性为“MyModify”和“MyDelete”。打开 GridView1 的属性窗口，设置 DataKeyNames 为“classID”、“courseID”、“teaID”和“team”。

（12）切换到“源”视图中，在 ID 为“Result”的 DIV 标记的下部添加一个 ID 为“Operation”的DIV。选中此DIV，在“应用样式”窗口中，单击“innerLayout”样式规则，

为该层应用样式规则。

（13）切换到设计视图，把光标定位到 Operation 层的内部，插入一个 Panel，ID 设置为“PanelModify”，再在 Panel 内插入一个 5 行 3 列的表格，指定宽度为“100%”。选中表格的第一列，通过“应用样式”为其应用“ThreeTableOne”样式；选中表格的第二列，为其应用“ThreeTableTwo”样式；选中表格第三列，为其应用“ThreeTableThree”样式。

（14）按照如图 6.30 所示在表格内添加控件。

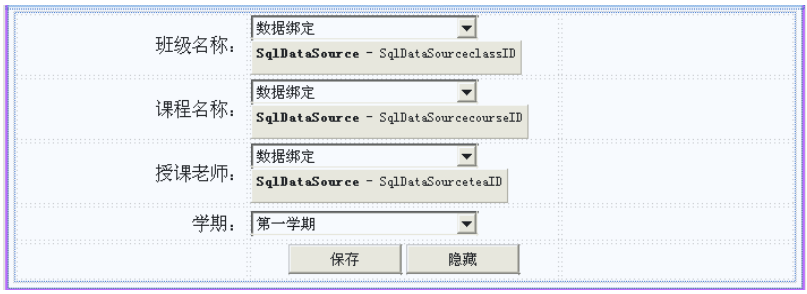


图 6.30　控件布局

从上到下分别设置 DIV 内控件的属性，设置如表 6.9 所示。

表 6.9　控件属性设置

控 件 类 型	相关属性和值
DropDownList	ID=“ DropDownListclassID”，width=“200 px”， DataSourceID=“SqlDataSourceclassID”,DataTextFiled=“className”， DataValueField=“classID”
SqlDataSource	ID=“SqlDataSourceclassID”， ConnectionString=“SqlServer2005ConnectionString”， SelectQuery=“ SELECT [classID], [className] FROM [Class]”
DropDownList	ID=“ DropDownListcourseID”，width=“200 px” DataSourceID=“SqlDataSourcecourseID”， DataTextFiled=“courseName”,DataValueField=“courseID”
SqlDataSource	ID=“SqlDataSourcecourseID”， ConnectionString=“SqlServer2005ConnectionString”， SelectQuery=“ SELECT [courseID], [courseName] FROM [Course]”
DropDownList	ID=“ DropDownListteaID”，width=“200 px”， DataSourceID=“SqlDataSourceteaID”， DataTextFiled=“teaName”,DataValueField=“teaID”
SqlDataSource	ID=“SqlDataSourceteaID”， ConnectionString=“ SqlServer2005ConnectionString”， SelectQuery=“ SELECT [teaID], [teaName] FROM [Teacher] WHERE ([teaType] = '1')”
DropDownList	ID=“DropDownListteam”，width=“200 px” Items 内添加 6 项，分别设置 Text 为“第一学期”~“第八学期”，Valuse 为“1”~“8”
Button	ID=“ButtonSave”，width=“100 px”，Text=“保存”
Button	ID=“ButtonHidden”，width=“100 px”，Text=“保存”

3. 编写本页所需要的代码

（1）本页的.aspx.cs 页面内添加如下的命名空间引用。

```
using System.Data.SqlClient;
using System.Collections.Generic;
```

（2）为了能让页面第一次访问的时候下部编辑区域不显示，当用户单击“排课”按钮的时候自动显示出来，需要在 Page_load 方法内添加如下的代码：

```
if(Session["userType"].ToString() != "0")                                    //如果不是管理员则返回
{    Response.Write("<script language=javascript>history.go(-1);</script>");  }     //重定向到前面页面
if (!Page.IsPostBack)
{    this.PanelModify.Visible = false;  }
```

（3）当用户单击“排课”按钮的时候，显示出 PanalModify 内部的内容，并保存当前操作状态为“add”状态。双击“排课”按钮，在 ButtonAdd_Click 方法中添加如下代码：

```
this.PanelModify.Visible = true;
ViewState["op"] = "add";
```

（4）为了使页面中的 DropDownList 控件根据某个选项的值而使此值对应的选项被选中，在 commTools 类中添加一个方法实现此功能，在需要的地方调用此类的方法即可。代码如下所示：

```
public static void setDropDownListByValue(DropDownList list, string value)
{
    for (int i = 0; i < list.Items.Count; i++)
    {
        if (list.Items[i].Value == value)
            list.Items[i].Selected = true;
        else
            list.Items[i].Selected = false;
    }
}
```

（5）当单击 GridView1 内部的“编辑”按钮的时候，能够获取相应行的信息，并自动显示出编辑视图，让用户进行编辑，需要在 GridView1 的 RowCommand 事件内添加如下的代码：

```
//判断是否单击的编辑按钮，编辑按钮的 CommandName 属性为“MyModify”
if (e.CommandName == "MyModify")
 {
    //获取操作行的索引
    int index = Convert.ToInt32(e.CommandArgument);
    //利用 DataKeys 获取操作行主键的值
    string classID = this.GridView1.DataKeys[index]["classID"].ToString();
    string courseID = this.GridView1.DataKeys[index]["courseID"].ToString();
    string team = this.GridView1.DataKeys[index]["team"].ToString();
```

```
        string teaID = this.GridView1.DataKeys[index]["teaID"].ToString();
        //保存主键的值，以便修改的时候使用
        ViewState["classID"] = classID;
        ViewState["courseID"] = courseID;
        ViewState["team"] = team;
        ViewState["teaID"] = teaID;
        //显示修改 Panel，并且设置对应的项选中
        this.PanelModify.Visible = true;
        commTools.setDropDownListByValue(this.DropDownListclassID, classID);
        commTools.setDropDownListByValue(this.DropDownListcourseID, courseID);
        commTools.setDropDownListByValue(this.DropDownListteam, team);
        commTools.setDropDownListByValue(this.DropDownListteaID, teaID);
        //保存操作状态
        ViewState["op"] = "modify";
    }
```

（6）当单击“GridView”内部的“删除”按钮的时候，能够获取相应行的信息，并从数据库内把此记录删除，需要在 GridView1 的 RowCommand 事件内添加如下的代码：

```
    //判断是否单击的删除按钮，删除按钮的 CommandName 属性为“MyDelete”
    if (e.CommandName == "MyDelete")
     {
        //获取操作行的索引
        int index = Convert.ToInt32(e.CommandArgument);
        //利用 DataKeys 获取操作行主键的值
        string classID = this.GridView1.DataKeys[index]["classID"].ToString();
        string courseID = this.GridView1.DataKeys[index]["courseID"].ToString();
        string team = this.GridView1.DataKeys[index]["team"].ToString();
        string teaID = this.GridView1.DataKeys[index]["teaID"].ToString();
        //创建 Sql 语句和参数，调用数据访问层的方法删除记录
        string sql = @"DELETE FROM Arrange
        WHERE classID=@classID AND courseID =@courseID AND team=@team AND teaID=@teaID";
        List<SqlParameter> para = new List<SqlParameter>();
        para.Add(commDBHelper.CreateParameters("@classID",DbType.String,classID));
        para.Add(commDBHelper.CreateParameters("@courseID", DbType.Int32, courseID));
        para.Add(commDBHelper.CreateParameters("@team", DbType.String, team));
        para.Add(commDBHelper.CreateParameters("teaID", DbType.Int32, teaID));
        if (commDBHelper.ExecuteNonQuery(sql, CommandType.Text, para))
        {
            commTools.alert(this, "删除成功");
            this.GridView1.DataBind();
        }
```

```
        else
            commTools.alert(this, "删除失败");
    }
```

（7）在新增或编辑视图下，单击“保存”（ButtonSave）按钮，则会根据当前的操作状态执行不同的操作，因此需要在 ButtonSave 的 Click 事件中添加如下的代码：

```
if (ViewState["op"].ToString() == "add")
 {
    string sql = @"INSERT INTO Arrange(classID, courseID, isRecord, team, teaID)
                    VALUES(@classID,@courseID,@isRecord,@team,@teaID)";
    List<SqlParameter> para = new List<SqlParameter>();
    para.Add(commDBHelper.CreateParameters("@classID", DbType.String,
            this.DropDownListclassID.SelectedValue));
    para.Add(commDBHelper.CreateParameters("@courseID", DbType.Int32,
            this.DropDownListcourseID.SelectedValue));
    para.Add(commDBHelper.CreateParameters("@isRecord", DbType.String, "0"));
    para.Add(commDBHelper.CreateParameters("@team", DbType.String,
            this.DropDownListteam.SelectedValue));
    para.Add(commDBHelper.CreateParameters("@teaID", DbType.Int32,
            this.DropDownListteaID.SelectedValue));
    if (commDBHelper.ExecuteNonQuery(sql, CommandType.Text, para))
    {
        commTools.alert(this, "保存成功");
        this.GridView1.DataBind();
        this.PanelModify.Visible = false;
    }
    else
        commTools.alert(this, "保存失败");
}
 if (ViewState["op"].ToString() == "modify")
 {
    string sql = @"UPDATE Arrange SET classID = @classID, courseID = @courseID,
            team = @team, teaID = @teaID
            WHERE (classID = @oldclassID) AND (courseID = @oldcourseID)
            AND (team = @oldteam) AND (teaID = @oldteaID)";
    List<SqlParameter> para = new List<SqlParameter>();
    para.Add(commDBHelper.CreateParameters("@classID", DbType.String,
            this.DropDownListclassID.SelectedValue));
    para.Add(commDBHelper.CreateParameters("@courseID", DbType.Int32,
          this.DropDownListcourseID.SelectedValue));
    para.Add(commDBHelper.CreateParameters("@team", DbType.String,
```

```
            this.DropDownListteam.SelectedValue));
        para.Add(commDBHelper.CreateParameters("@teaID", DbType.Int32,
            this.DropDownListteaID.SelectedValue));
        para.Add(commDBHelper.CreateParameters("@oldclassID", DbType.String,
            ViewState["classID"].ToString()));
        para.Add(commDBHelper.CreateParameters("@oldcourseID", DbType.Int32,
            ViewState["courseID"].ToString()));
        para.Add(commDBHelper.CreateParameters("@oldteam", DbType.String,
            ViewState["team"].ToString()));
        para.Add(commDBHelper.CreateParameters("@oldteaID", DbType.Int32,
            ViewState["teaID"].ToString()));
        if (commDBHelper.ExecuteNonQuery(sql, CommandType.Text, para))
        {
            commTools.alert(this, "修改成功");
            this.GridView1.DataBind();
            this.PanelModify.Visible = false;
        }
        else
            commTools.alert(this, "修改失败");
    }
```

（8）“隐藏”按钮是隐藏实现编辑课程安排功能的控件，即隐藏ID为“PanelModify”的Panel控件。此按钮的Click事件代码如下所示。

```
this.PanelModify.Visible = false;
```

4．在浏览器中使用本页

（1）在“设计视图”中，右击选择“在浏览器中查看”，出现如图6.31所示的界面。

图6.31　排课页面

（2）单击“排课”按钮，出现如图6.32所示的界面，选择相应的信息后，单击“保存”按钮即可进行排课。

图 6.32　新增界面

（3）单击 GridView 内部的“编辑”按钮，也会出现如图 6.32 所示的界面，可以更改排课信息。

第二部分　知识点链接

L6.1　数据源控件

L1. 数据源控件概述

ASP.NET 应用程序可以通过 ADO.NET 访问和操作数据库，但是 ADO.NET 对于程序员的要求门槛比较高，需要了解和熟悉 ADO.NET 的对象模型才可以方便地操作数据，而且使用起来比较繁琐。为此，微软提供了数据源控件，通过数据源控件，程序员只需编写很少的代码就可以很方便地操作数据库。

数据源控件对操作数据库方面的改观在于：

① 无须了解 ADO.NET 的细节，只要知道 Select、Insert、Update、Delete 语法即可。

② 避免了人为撰写的性能差的 ADO.NET 语法，提高数据访问的性能。

③ 与数据绑定控件结合，不需要代码或极少代码即可使用内置的分页、排序、编辑、更新和删除等功能。

④ 对不同的数据源提供了一致的程序访问模型，简化了开发和学习不同技术的负担。

⑤ 内置缓存机制，可以加快数据访问。

数据源控件可以连接不同类型的数据源，如数据库、XML 文档、其他对象等，但它留给设计者的接口却非常相似。设计人员只需采用相同或相似的方法处理数据，而不必关心数据源属于什么类型。.NET Framework 包含支持不同数据绑定方案的数据源控件。表 6.10 描述了内置的数据源控件。

表 6.10 .NET Framework 包含的数据源控件

数据源控件	说 明
SqlDataSource	允许使用 Microsoft SQL Server、OLE DB、ODBC 或 Oracle 数据库。与 SQL Server 一起使用时支持高级缓存功能
ObjectDataSource	允许使用业务对象或其他类，以及创建依赖中间层对象管理数据的 Web 应用程序。支持对其他数据源控件不可用的高级排序和分页方案
AccessDataSource	允许使用 Microsoft Access 数据库。当数据作为 DataSet 对象返回时，支持排序、筛选和分页
XmlDataSource	允许使用 XML 文件，特别适用于分层的 ASP.NET 服务器控件，如 TreeView 或 Menu 控件
SiteMapDataSource	结合 ASP.NET 站点导航使用

下面分别介绍每种类型的数据源控件。

1．SqlDataSource数据源控件

通过 SqlDataSource 控件，可以使用 Web 控件访问位于某个关系数据库中的数据，该数据库包括Microsoft SQL Server和Oracle数据库，以及 OLE DB 和 ODBC 数据源。将 SqlDataSource 控件和数据绑定控件结合使用，使用很少的代码或不使用代码就可以在 ASP.NET 网页中显示和操作数据。

2．ObjectDataSource 数据源控件

ObjectDataSource 数据源控件用于向数据绑定控件表示识别数据的中间层对象或数据接口对象。结合使用 ObjectDataSource 控件与数据绑定控件，只用少量代码或不用代码就可以在网页上显示、编辑和排序数据。

3．AccessDataSource数据源控件

AccessDataSource 是使用 Microsoft Access 数据库的数据源控件。与 SqlDataSource 一样，AccessDataSource 控件使用 SQL 查询执行数据检索。

AccessDataSource 控件的一个独特之处是不用设置 ConnectionString 属性。需要做的就是使用 DataFile 属性设置 Access 数据库的 mdb 文件位置，AccessDataSource 将负责维护数据库的基础连接。

4．XmlDataSource数据源控件

XmlDataSource 控件使得 XML 数据可用于数据绑定控件。XmlDataSource 从使用 DataFile 属性指定的 XML 文件加载 XML 数据。另外，还可以从使用 Data 属性的字符串加载 XML 数据。

5．SiteMapDataSource数据源控件

SiteMapDataSource 控件是站点地图数据的数据源，站点数据则由为站点配置的站点地图提供程序进行存储。SiteMapDataSource 使那些并非专门作为站点导航控件的 Web 服务器控件（如 TreeView、Menu 和 DropDownList 控件）能够绑定到分层的站点地图数据。可以使用这些 Web 服务器控件将站点地图显示为目录，或者对站点进行主动式导航。

L2. SqlDataSource

SqlDataSource 数据源控件用于访问 SQL 关系数据库中的数据。SqlDataSource 控件可以与其他数据绑定控件一起使用，开发人员用极少代码甚至不用代码，就可以在 ASP.NET 网页上显示和操作数据库。

SqlDataSource 数据源控件主要提供了如下的功能。

① 只需少量代码即可实现数据库操作：查询、插入、更新、删除。

② 以 DataReader 和 DataSet 方式返回查询结果集。

③ 提供缓存功能。

④ 提供冲突检测功能。

SqlDataSource 控件的主要属性如表 6.11 所示。

表 6.11　SqlDataSource 控件的主要属性

名　称	说　明
ConnectionString	获取或设置数据源控件的数据源连接字符串
ProviderName	获取或设置.NET FrameWork 数据提供程序的名称
SelectCommand	获取或设置数据源控件从数据源中检索数据所使用的 SQL 字符串
SelectCommandType	获取或设置 SelectCommand 属性中的文本是 SQL 语句，还是存储过程名称
SelectParameters	获取或设置 SelectCommand 属性中 SQL 字符串所使用的参数集合
UpdateCommand	获取或设置数据源控件向数据源中更新数据所使用的 SQL 字符串
UpdateCommandType	获取或设置 UpdateCommand 属性中的文本是 SQL 语句，还是存储过程名称
UpdateParameters	获取或设置 UpdateCommand 属性中 SQL 字符串所使用的参数集合
InsertCommand	获取或设置数据源控件向数据源中插入数据所使用的 SQL 字符串
InsertCommandType	获取或设置 InsertCommand 属性中的文本是 SQL 语句，还是存储过程名称
InsertParameters	获取或设置 InsertCommond 属性中 SQL 字符串所使用的参数集合
DeleteCommand	获取或设置数据源控件从数据源中删除数据所使用的 SQL 字符串
DeleteCommandType	获取或设置 DeleteCommand 属性中的文本是 SQL 语句，还是存储过程名称
DeleteParameters	获取或设置 DeleteCommand 属性中 SQL 字符串所使用的参数集合
EnableCaching	获取或设置一个值，该值指示 SqlDataSource 控件是否启用数据缓存

使用 SqlDataSource 的常见方法：

① 通过图形化的方式使用 SqlDataSource 实现数据的查询功能。

② 通过图形化的方式使用 SqlDataSource 实现数据的操作功能。

③ 通过编程的方式使用 SqlDataSource 实现数据的操作功能。

【例 6-1】　通过图形化方式使用 SqlDataSource 实现数据的查询功能。

（1）运行 Visual Studio 2008，新建站点，并命名为“数据源与数据绑定控件”，并把数据库“SMS”附加到 SQL Server 2005 中。

（2）新建 Web 页面“图形化使用 SqlDataSource 控件实现数据查询功能.aspx”，切换到“设计”视图。

（3）从工具箱中分别拖放一个 SqlDataSource 控件、一个 GridView 控件、一个 TextBox 控件和一个 Button 控件到此页面上，单击“SqlDataSource 任务”中的“配置数据源”，在弹

出的窗口中单击“新建连接”按钮，弹出“添加连接”对话框，在“服务器名”处选择所要连接的数据库名称，“登录到服务器”处选择登录服务器的方式，“连接到一个数据库”处选择所要连接的数据库，如图 6.33 所示。

图 6.33　配置数据库连接

（4）确定后，单击“下一步”按钮，会提示是否把此连接保存到Web.config 中，以便以后使用，单击“下一步”按钮，弹出“配置 Select 语句”对话框，如图 6.34 所示。

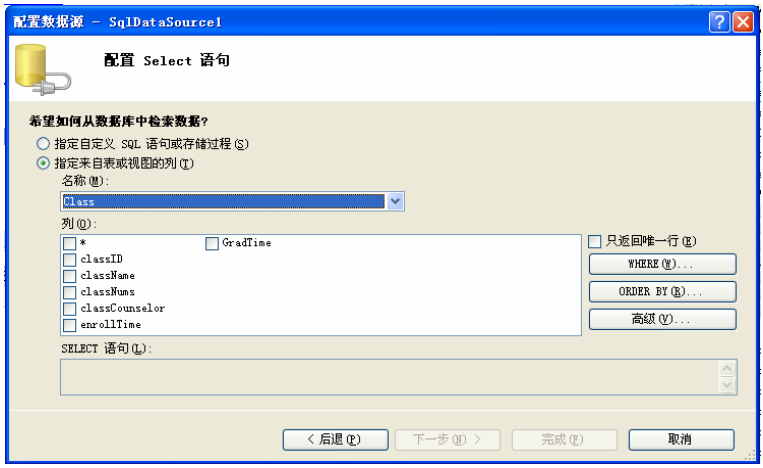


图 6.34　配置 Select 语句

（5）此对话框中提供两种设置 SQL 语句的选项：一种是“指定自定义 SQL 语句或存储过程”，此选项可以为数据源控件自定义 SQL 语句或存储过程，此种情况下适合于查询语句比较复杂的多表查询或者调用存储过程的情况，也可以同时为 SqlDataSource 配置其他的 Command；另一种是“指定来自表或视图的列”（此种情况适合于从单表查询数据）。

（6）选择第一种后，出现如图 6.35 所示的界面，此界面包括 4 个选项卡，可以分别为

数据源控件添加 SELECT、UPDATE、INSERT 和 DELETE 等 SQL 语句，或指定存储过程，SQL 语句中可以包含参数。

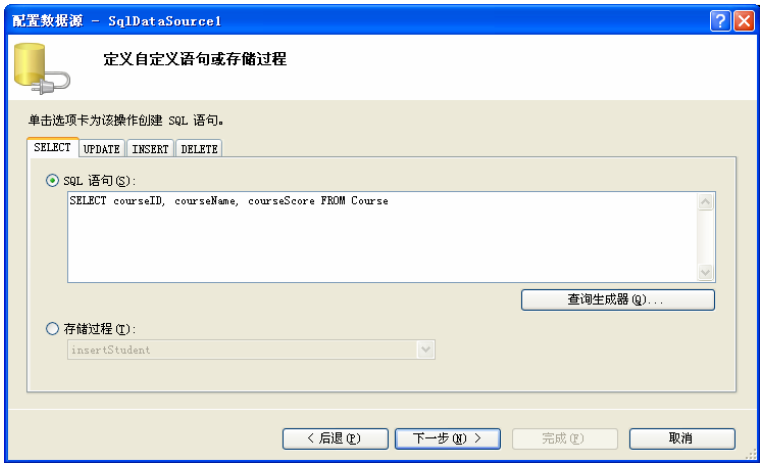


图 6.35　自定义语句或存储过程

（7）单击“下一步”按钮，如果 SQL 语句或存储过程中包含参数，则会弹出“定义参数”对话框，如图 6.36 所示。为 courseScore 参数指定参数源类型为 Control，ControlID 为“TextBox1”，设置完毕后，单击“下一步”、“完成”按钮完成数据源的创建。

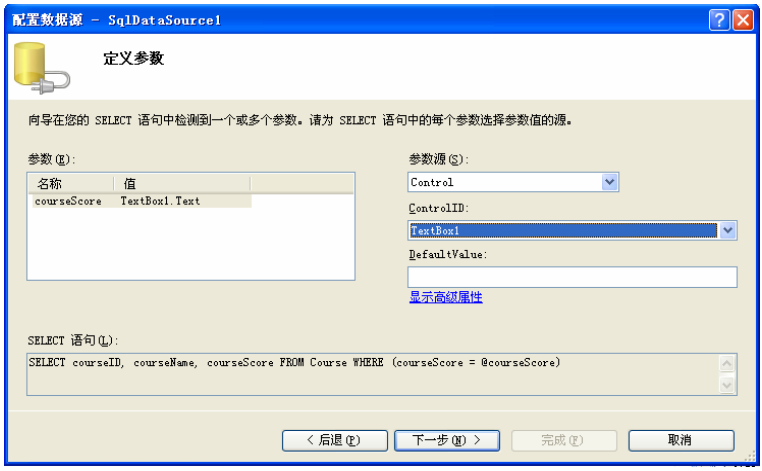


图 6.36　定义参数

SqlDataSource 参数源支持如下几种参数源，通过参数源可以方便地利用 ASP.NET 内置的对象或控件给 Sql 语句中的参数进行赋值：

Cookie：把 Cookie 对象中的某个变量的值作为参数的值。

Control：把服务器控件的某个属性的值作为参数的值。

Form：把 Form 表单内的元素的值作为参数的值。

Profile：把 Profile 文件中的属性的值作为参数的值。

QueryString：把 QueryString 查询字符串中的变量的值作为参数的值。

Session：把 Session 对象中的变量的值作为参数的值。

（8）设置 GridView 的数据源为 SqlDataSource1，在浏览器中查看本页，在文本框内输入对应的学分，单击 Button 控件，则 GridView 控件中就会显示对应的课程，运行效果如图 6.37 所示。

注：数据源控件绑定到数据绑定控件以后，数据绑定控件会自动调用数据源控件内的 SELECT 语句，无须自动调用，但是对于别的数据修改语句 UPDATE、DELETE、INSERT，则需要通过手动调用才会执行。

【例 6-2】通过图形化的方式和编程的方式使用 SqlDataSource 实现数据的操作功能。

（1）运行 Visual Studio 2008，打开站点“数据源与数据绑定控件”，在站点内新建页面，命名为“使用 SqlDataSource 实现数据的操作功能.aspx”。

（2）按照如图 6.38 所示设计界面。

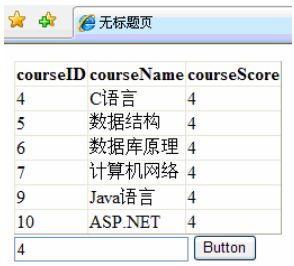


图 6.37　运行效果

图 6.38　页面布局

（3）打开 SqlDataSource1 的属性窗口，设置 ConnectionString 为“SMS-ConnectionString”，打开 InsertQuery 设置窗口，设置 Insert 命令为“INSERT INTO Course(courseName, courseScore) VALUES (@courseName, @courseScore)”，设置@courseName 的参数源为“TextBox1”，设置@courseScore 的参数源为“TextBox2”。设置完成后如图 6.39 所示。

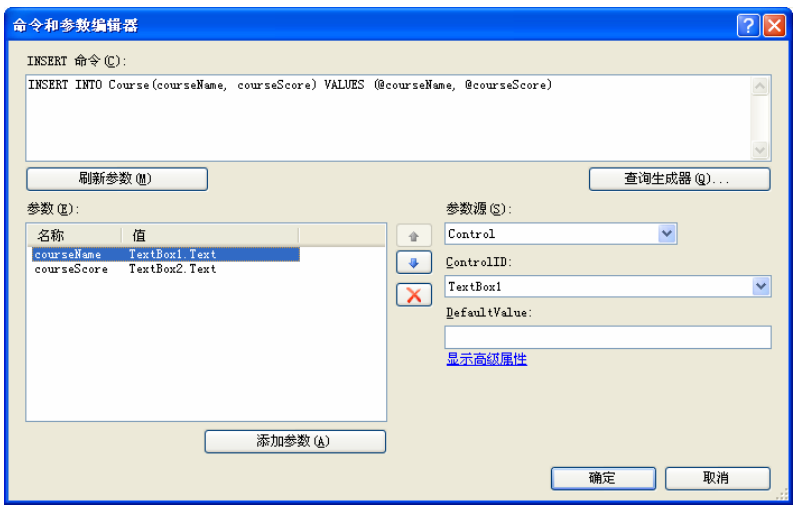


图 6.39　Insert 命令设置

（4）双击“图形方式”按钮，在 Click 事件内添加如下的代码：

```
try
{
    this.SqlDataSource1.Insert();
    //在页面上注册一段 JavaScript 脚本来显示提示框
    string script = "alert('插入成功')";
    Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "success", script, true);
}
catch
{
    string script = "alert('插入失败')";
    Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "success", script, true);
}
```

（5）在浏览器中查看本页，并录入相应的课程名称与课程学分，单击“图形录入”按钮，即可把录入的数据插入到数据库内。

（6）以同样的方式设置 SqlDataSource2 的 ConnectionString 属性和 Insert 命令，但不设置参数源，双击“编程方式”按钮，在 Click 事件内添加如下的代码来实现通过代码给参数赋值：

```
try
{
    //通过代码给 SqlDataSource2 中 Insert 命令所用的参数赋值
    this.SqlDataSource2.InsertParameters["courseName"].DefaultValue = this.TextBox1.Text;
    this.SqlDataSource2.InsertParameters["courseScore"].DefaultValue = this.TextBox2.Text;
    this.SqlDataSource2.Insert();
    string script = "alert('插入成功')";
    Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "success", script, true);
}
catch
{
    string script = "alert('插入失败')";
    Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "success", script, true);
}
```

（7）在浏览器中查看本页，在文本框内录入相应的课程名称与课程学分，单击“编程方式”按钮实现把录入的数据插入到数据库中。

L3. ObjectDataSource

在实际的信息系统开发过程中，有些业务逻辑很复杂，不是简单的几条 SQL 语句就可以解决的问题，这时就需要采用功能灵活的 ObjectDataSource 数据源控件来构建多层的系统开发。

在复杂的多层系统中，数据库的操作通常被封装为一个业务逻辑类，在业务逻辑类内

创建一个方法，在方法内编写代码实现数据操作功能。但通过数据源控件能够和数据绑定控件很好地结合，无须代码就可以实现排序、分页等复杂的功能，因此在此种情况下通过把业务逻辑类和 ObjectDataSource 进行绑定，ObjectDataSource 不直接去操作数据库，而是通过调用业务逻辑类内的方法来实现数据的查询和修改操作。

【例 6-3】利用 ObjectDataSource 操作数据库。

（1）运行 Visual Studio 2008，打开“数据源与数据绑定控件”站点，并添加 ObjectDataSource.aspx 页面。

（2）通过添加新项，选择“类”并命名为“DataManager.cs”，当单击“添加”按钮时提示是否把类存放到 App_Code 文件夹，单击“是”按钮。

（3）在 DataManager.cs 中添加命名空间，命名空间如下所示：

```
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
```

为 DataManager 类添加代码，代码如下所示，添加结束后，选择“生成”→“生成网站”菜单项，确保代码生成成功。

```
public class DataManager
{
    public DataManager()
    {
        // TODO: 在此处添加构造函数逻辑
    }
    public DataSet getData(string value)
    {
        if (value != null)
        {
            SqlConnection conn = new SqlConnection();
            conn.ConnectionString=ConfigurationManager.ConnectionStrings["SMSConnectionString"].
                ConnectionString;
            conn.Open();
            string sql = " SELECT courseID, courseName, courseScore FROM Course WHERE
                (courseScore = @courseScore)";
            SqlCommand comm = new SqlCommand(sql, conn);
            comm.Parameters.AddWithValue("courseScore", value);
            DataSet dataset = new DataSet();
            SqlDataAdapter adapter = new SqlDataAdapter(comm);
            adapter.Fill(dataset);
            comm.Dispose();
            conn.Close();
            return dataset;
```

```
            }
            return null;
        }
    }
```

方法的参数类似于SQL语句中的参数，当跟 ObjectDataSource 数据源控件进行绑定的时候，可以通过 ObjectDataSource 数据源控件给方法参数传值，在方法内部可以利用传递的参数进行数据查询或处理工作。本例只是演示如何通过 ObjectDataSource 跟业务逻辑类的方法进行绑定，并传递参数，方法内部并没有进行复杂的数据处理工作。

（4）从工具箱中拖放 ObjectDataSource 控件、GridView 控件、TextBox 控件和 Button 控件到 ObjectDataSource.aspx 页面内。

（5）单击 ObjectDataSource 任务，配置数据源，出现“选择业务对象”窗口，如图 6.40 所示，选择已经创建的 DataManager 类，单击“下一步”按钮。

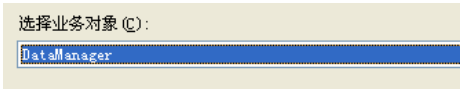


图 6.40　选择业务对象

（6）在弹出的“定义数据方法”对话框中选择“getData”方法，如图 6.41 所示。此对话框类似于 SqlDataSource 的“自定义语句或存储过程”对话框，设置数据源控件各种操作与业务对象内部方法的对应关系，当调用数据源控件内对应的方法时，数据源控件会自动调用业务逻辑类对应的方法，单击“下一步”按钮。

（7）如果方法需要输入参数，则会弹出“定义参数”对话框，等同于 SqlDataSource 控件，把 TextBox1 的 Text 属性设置为参数的值，单击“完成”按钮完成数据源的配置。

（8）把 GridView1 绑定到 OjbectDataSource1 控件上，在浏览器内查看本页，输入“4”并单击“Button”按钮，运行结果如图 6.42 所示。

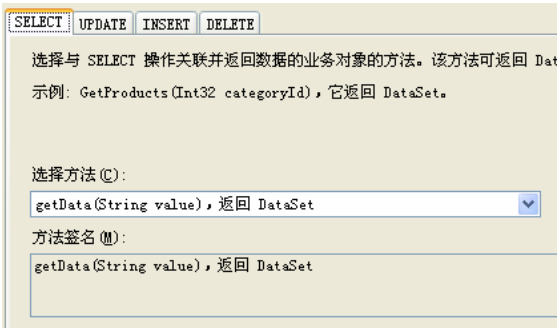


图 6.41　定义数据方法

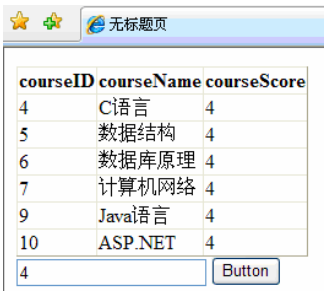


图 6.42　定义数据方法

L6.2　数据绑定控件

L1. 数据绑定的概念

数据绑定（Data Binding）是一项非常简单、有效的技术。ASP.NET 采用数据绑定技术

将显示控件的某个属性与数据源绑定在一起。当数据源中的数据发生变化且重新请求网页时，被绑定对象中的属性将随数据源而改变。

数据绑定应用的范围非常广泛，数据集、数组、集合或 XML 文档，甚至一般的变量，都可以作为数据源，大多数控件的属性都可以成为被绑定的对象。在 ASP.NET 的类库中，所有的绑定控件都从 BaseDataBoundControl 基类继承，因此具有很多相似的功能。然后，不同类型的绑定控件继承于不同的子类。类的层次关系如图 6.43 所示。

几个基类如下:

① BaseDataBoundControl 类：所有绑定控件的基类。

② DataBoundControl 类：所有常用控件的基类。

③ ListControl 类：所有列表控件的基类。

④ CompositeDataBoundControl 类：比较复杂的表格控件的基类。

⑤ HierarchicalDataBoundControl 类：所有层次控件的基类。

根据绑定控件的功能特性可以把数据绑定控件分为如下几类：

① 列表绑定控件：此类主要包括 DropDownList、ListBox、RadioButtonList 等，此类控件的使用方法相同。

② 复合绑定控件：此类控件主要包括 GridView、DetailsView、ListView 等，不同的控件使用方法大致相同，但是不同的控件具有不同的功能。

③ 层次绑定控件：此类控件主要包括 Menu、TreeView 控件。

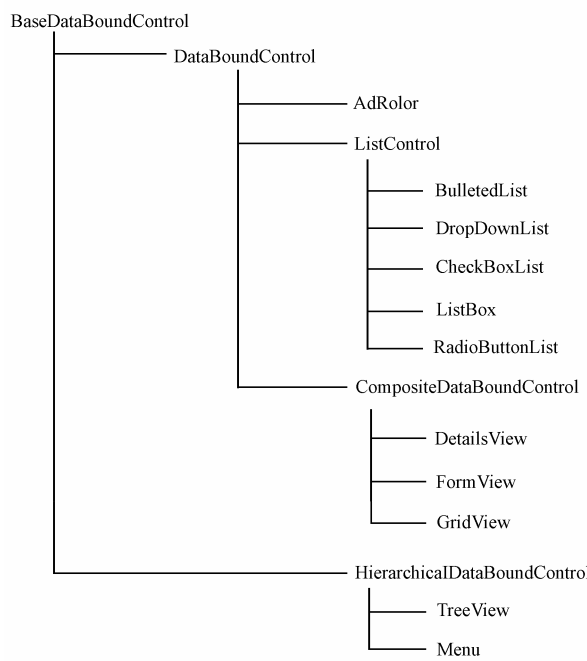


图 6.43　数据绑定控件的类层次结构

L2. DropDownList控件

DropDownList 服务器控件是常用的数据绑定控件，主要用来提供单条数据的选择功能。DropDownList 除了前面介绍的通过手工方式给 DropDownList 添加项外，还提供了数据绑定的功能，可以方便地与数据源控件进行绑定，自动从数据源内获取数据来添加项。

表 6.12 列出了 DropDownList 与数据绑定有关的属性。

表 6.12　DropDownList 相关属性

名　称	说　明
DataSourceID	用来设置与 DropDownList 进行绑定的数据源控件的 ID
DataTextField	用来设置数据源中哪个字段跟 Item 的 Text 绑定
DataValueField	用来设置数据源中哪个字段跟 Item 的 Value 绑定

当 DropDownList 跟数据源控件进行绑定的时候，需要设置 DataSourceID 为数据源控件的 ID，设置 Text 属性跟数据源中某个字段进行绑定，设置 Value 属性跟数据源中某个字段进行绑定。运行时数据源内所查询出来的记录就会自动填充为 DropDownList 的项，并且每条记录中相应的字段的值会自动赋给 DropDownList 项的 Text 属性和 Value 属性。

L3. GridView控件

GridView 控件主要以表格的形式显示数据，并提供对列进行排序、分页、编辑或删除单个记录的功能。GridView 控件是 ASP.NET 的早期版本中提供的 DataGrid 控件的升级版控件。除了可以添加利用数据源控件的新功能以外，GridView 控件还实现了如定义多个主键字段、使用绑定字段和模板、允许自定义用户界面及用于处理或取消事件的新功能。

可以使用 GridView 来完成以下操作：

① 通过数据源控件自动绑定和显示数据。

② 通过数据源控件对数据进行选择、排序、分页、编辑和删除。

另外，还可以通过以下方式自定义 GridView 控件的外观和行为：

① 指定自定义列和样式。

② 利用模板创建自定义用户界面（UI）元素。

③ 通过处理事件将自己的代码添加到 GridView 控件的功能中。

1. 分页、排序和选择

虽然 GridView 配合数据源控件能正确、快速地显示数据，但是这还远远达不到实际应用的需要。例如，当数据记录较多时，用户可能需要分页浏览，而当显示商品列表时，用户又可能需要按照价格排序浏览。另外，单纯的显示记录还不够，很多场合需要用户编辑和删除这些记录等，这些功能在 GridView 中都得到较好的支持。下面将详细讲解这些内容。

（1）分页

在 ASP.NET 中 GridView 控件有内置的分页功能，可以使用默认分页用户界面或者创建自定义的分页界面。

GridView 控件支持对其数据源中的项进行分页。将 AllowPaging 属性设置为 True 可以启用分页。如果将 GridView 控件绑定到数据源控件上，则 GridView 控件将直接在界面级别

进行分页处理。这意味着 GridView 控件仅从数据源请求显示当前页所需的记录数，如果将 GridView 控件绑定到 Dataset 对象上，则分页功能需要编写代码完成。

可以通过多种方式自定义 GridView 控件的分页用户界面。通过使用 PageSize 属性来设置每一页显示的记录数，通过设置PageIndex 属性来设置 GridView 控件的当前页。还可以使用 PagerSettings 属性或通过提供页导航模板来指定更多的自定义行为，例如，通过设置 GridView 控件的 Mode 属性来自定义分页模式。

```
GridView1.PagerSettings.Mode = PagerButtons.NextPreviousFirstLast
```

PagerButtons 枚举表示分页导航的按钮类型，有以下 4 种可选项：

① PagerButtons.NextPrevious：上一页按钮和下一页按钮。

② PagerButtons.NextPreviousFirstLast：上一页按钮、下一页按钮、第一页按钮和最后一页按钮。

③ PagerButtons.Numeric：可直接访问页面的带编号的链接按钮。

④ PagerButtons.NumericFirstLast：带编号的链接按钮、第一个链接按钮和最后一个链接按钮。

为例 6-1 中的 GridView 控件启用分页，设置 PageSize 为 3，再次运行结果如图 6.44 所示。

（2）排序

GridView 控件提供了内置的排序功能，且无须任何编码。

在例 6-1 中将 GridView 控件的 AllowSorting 属性设置为 True，即可启用该控件中的默认排序行为，启用排序后，会自动为每列的 SortExpression 属性赋值（赋值为本字段所绑定的数据字段的名称）。启用排序后，标题栏会显示为超链接状态，表示可以进行排序了，如图 6.45 所示。

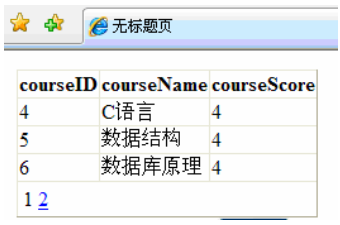


图 6.44 运行结果

图 6.45 为 GridView 启用内置排序

在运行时，用户可以单击某列标题中的LinkButton 控件对该列进行排序。单击该链接会使页面执行回发，并引发 GridView 控件的 Sorting 和 Sorted 事件。

当然，除了默认的排序功能之外，如果需要禁用对个别字段的排序，只需将列的 SortExpression 属性设置为空字符串（""）。

（3）选择

GridView 控件还有一个内置的选定内容功能，允许用户在网格中选择一行。在 GridView 控件中选择一行实际上不执行任何任务。但是，通过添加选定内容功能，可以向网格添加一些功能，在用户指向特定行时进行某种操作。例如，当用户选择某一行时，该行会以不同的外观重新显示；可以使用行选定内容来创建主/从报表方案，用户选择网格中显

示的主行，使详细信息记录显示在页上其他位置。

要启用选定内容的功能很简单：在设计视图中，单击 GridView 控件的智能标签，在智能标记面板中选择“启用选定内容”，此时 GridView 控件会多出一个选择列，如图 6.46 所示。

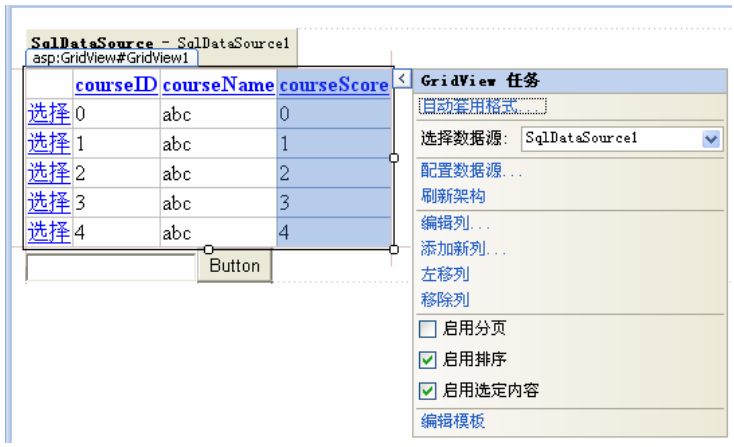


图 6.46　为 GridView 启用选定内容

当启用了“选定内容”功能后，可以通过设置 GridView 控件的一些属性来自定义外观样式，GridView 中提供了以下两种可以自定义选定行外观的功能。

① SelectedRowStyle 属性：例如，可以将 SelectedRowStyle 的 BackColor 子属性设置为灰色，选定的行则显示为具有灰色背景。

② 按照排序中介绍的方法，打开“字段”对话框，可以看到在启用了选定内容功能后，“选定的字段”中将多出一个“选择”列，选中它，可通过右边ButtonType 属性来更改按钮的样式，通过 SelectText 属性更改按钮文本，还可以通过 SelectImageUrl 属性将选择按钮替换为一幅图片，如图 6.47 所示。

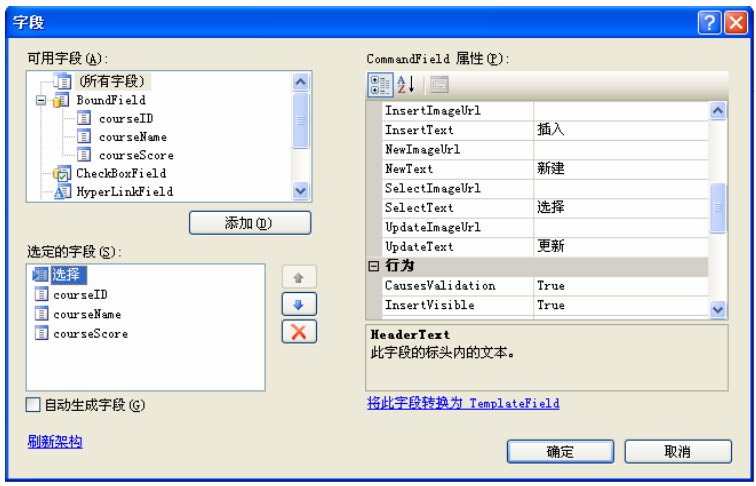


图 6.47　修改选择列的样式

在自定义了选择行的样式后，运行网页，效果如图 6.48 所示。

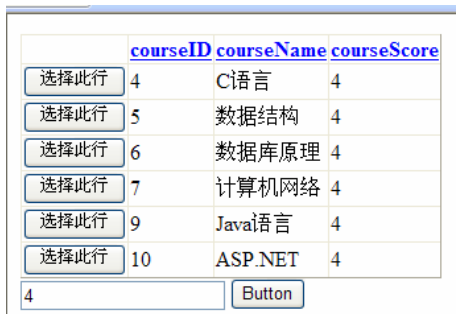


图 6.48　运行结果

2. 设置模板样式

模板（Template）是一组样板，它将 HTML 元素与ASP.NET 的控件结合在一起，用来定义数据的显示格式，并且由这些格式形成最终的布局。模板相当于框架，在框架中可以放入控件，通过控件与数据绑定，使得这些绑定的数据按照模板规定的格式显示。

模板与样式表（CSS）有联系但也有区别。样式表定义的是某个特定元素的属性（如该元素字体的类型、大小、颜色等）。而模板能够更加深刻地改变控件的整体外观，还能给控件增添新的功能。通常将模板与样式表结合起来，以便全面改善界面的显示，增强控件的功能。

控件中的模板由头、尾、体三部分组成，分别用头模板（HeaderTemplate）、尾模板（FooterTemplate）和体模板（ItemTemplate）表示。三部分的关系如图 6.49 所示。

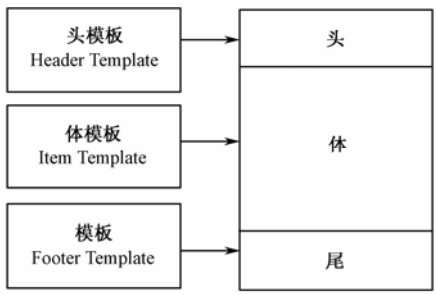


图 6.49　控件的模板结构

其中，头模板和尾模板用来设置数据标题和尾部显示的内容和格式，这两种模板是选用部分，而体模板则是必须选用的，它用来显示数据的主体。当绑定有多条记录时，在体模板中将自动扫描数据源的各条记录，并按照模板的要求逐条显示出来。体模板有时又可细分为选择模板、编辑模板等，用来定义被选中记录或编辑记录的显示样式。还可以用交替模板来设置交替记录的不同的样式（如不同的底色）。

控件通常具有固定的功能和显示界面。一旦控件拥有模板功能，就能在不同的情况下自动转换成不同的界面并执行不同的任务，控件的功能从而得到大大的加强，一个控件可以当多种控件来使用。

下面介绍在 GridView 控件中对模板的设置方法，GridView 控件中对模板的设置有两种方法。

（1）自动套用格式

这种方式设置起来最简单，但灵活性不够，功能也不够强。使用方法是：右击窗体页内 GridView 控件，在弹出的菜单中选择“自动套用格式”命令，弹出如图 6.50 所示的对话框。

图 6.50　自动套用格式

单击左边“选择架构”中的各项，右边的“预览”窗口中将显示出该方案所对应的显示界面。逐个单击左边的方案，直到选择一个合适的方案为止。最后单击“确定”按钮，完成模板的设置工作。

（2）设置模板样式

打开 GridView 控件的属性窗口，可以看到 6 个模板样式的选项：

① HeaderStyle：头模板样式。

② ItemStyle：体模板样式。

③ FooterStyle：尾模板样式。

④ AlternatingRowStyle：隔行模板样式。

⑤ SelectedItemStyle：选择项模板样式。

⑥ EditItemStyle：编辑项模板样式。

每种样式都包括：底色（BackColor）、边界颜色（BorderColor）、边界样式（BorderStyle）、边界宽度（BorderWidth）、其他样式（CssClass）等，利用这些属性可以定义模板的显示特征。

程序运行时，前 4 种模板的样式即可显示出来。而 SelectedItemStyle 和 EditItemStyle 模板只有当用鼠标选中某条记录或者对某条记录进行编辑时，才会显示出来。

例如，将隔行样式（AlternatingRowStyle）的背景设置成浅绿色时相应的代码如下：

```
<asp:GridView ID="GridView1" AutoGenerateColumns="False" Runat="server">
<AlternatingRowStyle BackColor="#C0FFFF"></AlternatingRowStyle>…</asp:GridView>
```

在设计视图中的效果如图 6.51 所示。

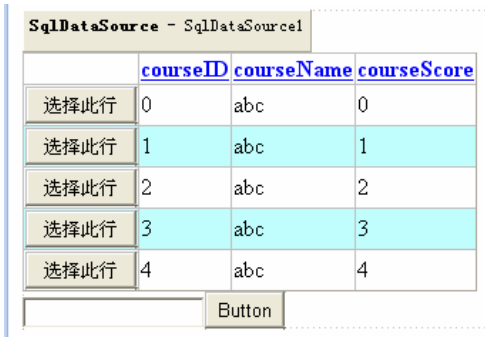


图 6.51　设计视图中的隔行样式

3. 更新数据表

GridView 控件具有一些内置功能，允许用户在不需要编程的情况下编辑或删除记录。可以使用 CommandField 和模板自定义 GridView 控件的编辑或删除功能。

启用更新数据表功能的具体步骤如下。

（1）通过数据源控件将 GridView 控件与数据库连接。

（2）配置数据源，当选择好数据表及相关字段后，单击“高级”按钮，界面如图 6.52 所示。

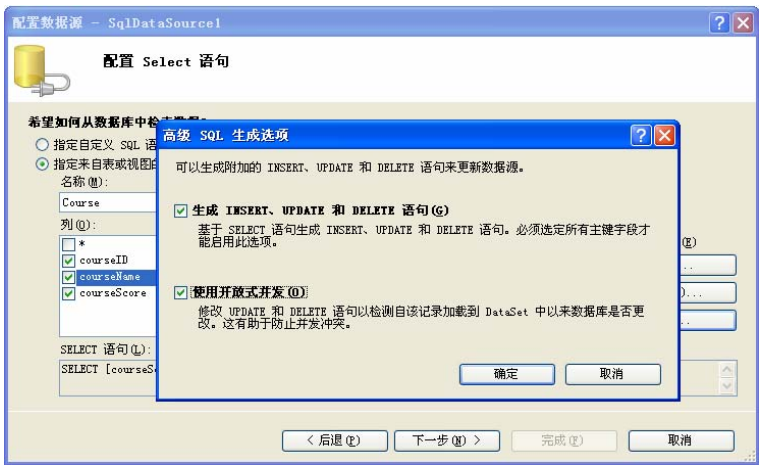


图 6.52　配置数据源的高级选项

这里提供了两个复选框。选中第一个复选框时，系统将自动产生插入（Insert）、更新（Update）和删除（Delete）的 SQL 语句；选择第二个复选框时，有助于防止由于同时对数据表进行更新和删除而引发的冲突。

（3）回到 GridView 控件并打开数据源控件，在智能标签面板中将看到启用编辑与启用删除等选项，如果选中这些复选框，系统将在 GridView 控件中显示出相应的按钮（如编辑、删除按钮等）。界面的显示情况如图 6.53 所示。

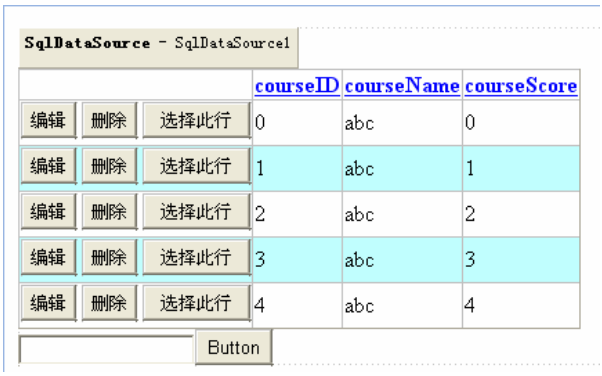


图 6.53　为 GridView 启用编辑和删除功能

现在打开“源”视图，可以看到系统不仅生成了插入、删除、更新数据表的 SQL 语句，同时还生成了参数赋值的语句，完整的代码如下：

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:SMSConnectionString %>"
    SelectCommand="SELECT [courseScore], [courseID], [courseName] FROM [Course] WHERE
        ([courseScore] = @courseScore)" ConflictDetection="CompareAllValues"
    DeleteCommand="DELETE FROM [Course] WHERE [courseID] = @original_courseID AND
        [courseScore] = @original_courseScore AND [courseName] = @original_courseName"
    InsertCommand="INSERT INTO [Course] ([courseScore], [courseName]) VALUES (@courseScore,
        @courseName)"
    OldValuesParameterFormatString="original_{0}"
    UpdateCommand="UPDATE [Course] SET [courseScore] = @courseScore, [courseName] =
     @courseName WHERE [courseID] = @original_courseID AND [courseScore] =
     @original_courseScore AND [courseName] = @original_courseName">
    <SelectParameters>
        <asp:ControlParameter ControlID="TextBox1" Name="courseScore"
            PropertyName="Text" Type="Int32" />
    </SelectParameters>
    <DeleteParameters>
        <asp:Parameter Name="original_courseID" Type="Int32" />
        <asp:Parameter Name="original_courseScore" Type="Int32" />
        <asp:Parameter Name="original_courseName" Type="String" />
    </DeleteParameters>
    <UpdateParameters>
        <asp:Parameter Name="courseScore" Type="Int32" />
        <asp:Parameter Name="courseName" Type="String" />
        <asp:Parameter Name="original_courseID" Type="Int32" />
        <asp:Parameter Name="original_courseScore" Type="Int32" />
        <asp:Parameter Name="original_courseName" Type="String" />
    </UpdateParameters>
```

```
        <InsertParameters>
            <asp:Parameter Name="courseScore" Type="Int32" />
            <asp:Parameter Name="courseName" Type="String" />
        </InsertParameters>
    </asp:SqlDataSource>
```

（4）运行程序，单击数据表格中的“编辑”按钮，各个字段的值出现在 TextBox 文本框中，方便用户更新数据，而原来的“编辑”也替换为“更新”、“取消”两个按钮，如图 6.54 所示。

图 6.54　编辑 GridView 中的数据

单击“更新”按钮，数据源控件将自动获取修改后的数据，并执行 UpdateCommand 中生成的 SQL 命令，以更新数据库中的记录。单击“取消”按钮将放弃此次更改，返回开始的显示界面。同样，也可以通过单击“删除”按钮直接删除记录。注意，当执行删除的时候，需要保证不能存在外键关联，如果要同时从多张表删除数据，需要自己编写代码实现。

此外，为了防止删除误操作，在删除操作前应总是向用户请求确认。可以为“删除”按钮添加客户端的脚本验证功能以实现删除前确认。其操作步骤如下。

（1）打开 GridView 控件的智能标签面板，选择“编辑列”，打开“字段”对话框，在“选定的字段”中选中CommandField，单击右下角的“将此字段转换为 TemplateField”，转换后如图 6.55 所示。

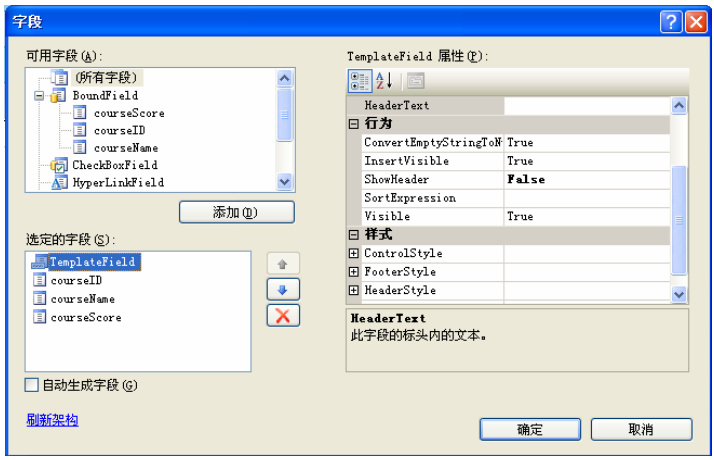


图 6.55　将 CommandField 转换为 TemplateField

转到“源”视图，对比一下转换前后的标记代码。转换前为：

```
<asp:CommandField ShowEditButton="True" />
```

转换后为：

```
<asp:TemplateField ShowHeader="False">
    <ItemTemplate>
        <asp:LinkButton ID="LinkButton1" runat="server" CausesValidation="False"
            CommandName="Delete"    Text="删除"></asp:LinkButton>
    </ItemTemplate>
 </asp:TemplateField>
```

可以看到，模板列具有更加复杂的结构和属性，毫无疑问，它也将拥有更多的可定制功能，后面将更详细地讲解模板列的使用。

（2）为数据行中的每个删除按钮添加脚本 OnClientClick 属性，代码如下：

```
<asp:TemplateField ShowHeader="False">
    <ItemTemplate>
        <asp:LinkButton ID="LinkButton1" OnClientClick="return confirm('确认删除此记录吗？');"
            runat="server" CausesValidation="False" CommandName="Delete" Text="删除">
        </asp:LinkButton>
    </ItemTemplate>
</asp:TemplateField>
```

（3）运行程序，单击任意数据行的“删除”按钮，弹出确认删除的消息框，单击“确定”将执行正常的删除操作，单击“取消”不会执行任何操作，直接返回，如图 6.56 所示。

图 6.56　删除前请求确认

4. 使用模板列

GridView 是由一组字段组成的，它们都指定了来自 DataSource 中的哪些属性需要用到呈现中。最简单的字段类型是 BoundField，它仅将数据简单地显示为文本。其他的字段类型使用交互HTML元素来显示数据。例如，CheckBoxField 将被呈现为一个 CheckBox，其选中状态由某特定数据字段的值来决定；ImageField 则将某特定数据字段呈现为一个图片，当然，这个数据字段中应该放的是图片类型的数据。超链接和按钮的状态取决于使用 HyperLinkField 或 ButtonField 字段类型的数据字段的值。

虽然 CheckBoxField、ImageField、HyperLinkField 和 ButtonField 考虑到了数据的交互视图，但它们仍然有一些相关的格式化的限制。为了适应用户对字段的个性化需求，

GridView 提供了使用模板来进行呈现的 TemplateField。模板可以包括静态的 HTML、Web 控件及数据绑定的代码。此外，TemplateField 还拥有各种可以用于不同情况的页面呈现的模板。例如，ItemTemplate 默认地用于呈现每行中的单元格，而 EditItemTemplate 则用于编辑数据时的自定义界面。

本节要使用命名为“XSCJ”数据库中的“XS”表，“XS”表的结构如表 6.13 所示。需要读者自己添加此数据库，并为“XS”表添加数据。

表 6.13　XS 表的表结构

项　目　名	列　　名	数 据 类 型	可　　空	默　认　值	说　　明
学号	XH	定长字符串型（char6）	×	无	主键，前 2 位年级，中间 2 位班级号，后 2 位序号
姓名	XM	定长字符串型（char8）	×	无	
性别	XB	位型（bit）	√	1	1：男；0：女
出生时间	CSSJ	日期型（datetime）	√	无	
专业	ZY	定长字符串型（char12）	√	无	
总学分	ZXF	整数型（int）	√	0	0≤总学分<160
备注	BZ	不定长字符串型（varchar500）	√	无	
联系方式	LXFS	扩展标记语言型（xml）	√	无	
照片	ZP	图像型（image）	√	无	

与前面不同的是，将使用一些TemplateField来自定义学生信息的呈现。特别地，将列出所有的学生，但将会把学号和姓名放在一列中，把他们的出生时间放在一个 Calendar 控件中，还将用一个附加列来显示他们的年龄。使用 TemplateField 来实现上述效果的具体步骤如下。

（1）将数据绑定到 GridView

从工具箱中拖一个 GridView 到设计器上。为其创建一个新的数据源控件 SqlDataSource1，配置数据源，具体过程在前文已经介绍过，其中配置 Select 语句的向导如图 6.57 所示。

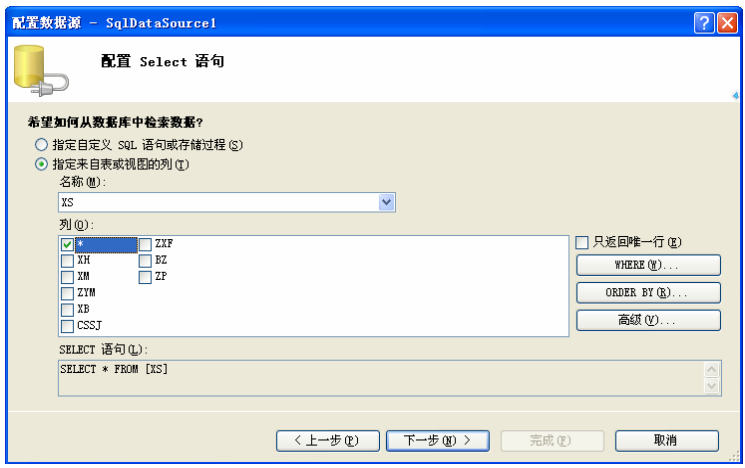


图 6.57　配置 Select 语句

在“XS”表中仅选取了[XH]、[XM]、[ZYM]、[XB]和[CSSJ] 这 5 个字段。数据源配置完成后，转到“源”视图，可以看到 GridView 的标记代码如下所示：

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False" DataKeyNames="XH"
    DataSourceID="SqlDataSource1">
    <Columns>
        <asp:BoundField DataField="XH" HeaderText="XH"ReadOnly="True"SortExpression="XH" />
        <asp:BoundField DataField="XM" HeaderText="XM" SortExpression="XM" />
        <asp:BoundField DataField="ZYM" HeaderText="ZYM" SortExpression="ZYM" />
        <asp:BoundField DataField="XB" HeaderText="XB" SortExpression="XB" />
        <asp:BoundField DataField="CSSJ" HeaderText="CSSJ" SortExpression="CSSJ" />
    </Columns>
</asp:GridView>
```

在进行下一步之前，通过编辑列，修改列名为“学号”、“姓名”、“专业名”、“性别”和“出生时间”，启用分页，每页5 条。先浏览一下目前的效果，运行程序，如图 6.58 所示，看到一个表格，表格中每一个记录都是一个学生的信息，一共有 5 列，分别是学号、姓名、专业名、性别和出生时间。

图 6.58　运行结果

（2）将学号和姓名显示在一列中

现在的效果是学号和姓名分开在两列中显示，而我们想要的效果是把它们放到一个列中，按照“学号：姓名”的格式显示出来。要做到这一点，需要用到 TemplateField。有两种方法实现，一是添加一个新的 TemplateField，给它加上一些必须的标记语言和数据绑定代码，然后删除原来的学号和姓名这两个 BoundField；二是将学号这个BoundField 直接转换成一个 TemplateField，编辑它以加上姓名的值，然后再删除姓名这个BoundField。这里采用第二种方法，直接转换可以节省很多工作。将 BoundField 转换成 TemplateField 的方法在前文也已经介绍过了，转换后的效果如图 6.59 所示。

更改之后，设计器中并没有什么明显的不同，这是因为将 BoundField 转换成 TemplateField 时，实际上是创建了一个维持之前的 BoundField 的外观的 TemplateField。尽管在设计器中没有视觉上的变化，但是这个转换的过程已经将 BoundField 的声明代码：

```
<asp:BoundField DataField="学号" HeaderText="学号" SortExpression="学号" />
```

改成了如下的 TemplateField 的声明代码：

```
<asp:TemplateField HeaderText="学号" SortExpression="学号">
<EditItemTemplate>
<asp:TextBox ID="TextBox1" runat="server" Text='<%# Bind("学号") %>'></asp:TextBox>
</EditItemTemplate>
<ItemTemplate>
<asp:Label ID="Label1" runat="server" Text='<%# Bind("学号") %>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>
```

图 6.59　将 BoundField 转换成 TemplateField

可以看到，TemplateField 由两个模板组成：一个 ItemTemplate，它有一个 Label 控件，其 Text 属性被设置为“学号”数据字段的值；还有一个 EditItemTemplate，它有一个 TextBox 控件，其 Text 属性也被设置为“学号”数据字段的值。数据绑定语法<%# Bind("学号") %>说明数据字段 FirstName 被绑定到了这个特定的 Web 控件的属性 Text 上。

要将姓名添加到 TemplateField 中，需要为 ItemTemplate 添加一个 Label 控件，并将其 Text 属性绑定到“姓名”字段上。通过设计器或是手工编写代码都可以做到这一点。单击“GridView 任务”中的“编辑列”，出现“模板编辑模式”窗口，如图 6.60 所示，从“显示”下拉列表中选中“学号”，编辑过后会出现如图 6.61 所示的界面。

在 ItemTemplate 模板内进行修改，改成如图 6.62 所示的效果，单击“Label 任务”，选中“编辑 DataBindings”，将出现如图 6.63 所示的界面，并把属性绑定到 XM 字段，单击“确定”按钮结束模板编辑。

接着通过编辑字段，删除“姓名”这个绑定列，并将“学号”这个模板列的列头文本（HeaderText）改成“学号及姓名”。修改后的效果如图 6.64 所示。

图 6.60　选择模板列图

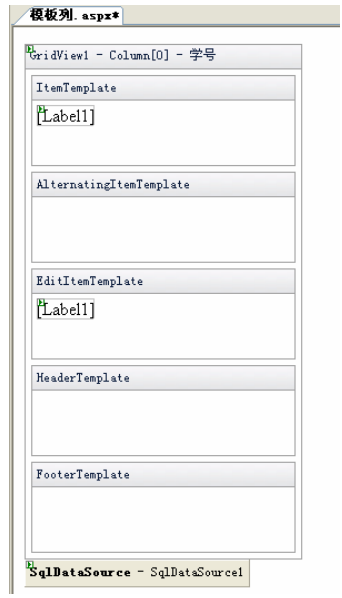


图 6.61　编辑模板

图 6.62　编辑模板

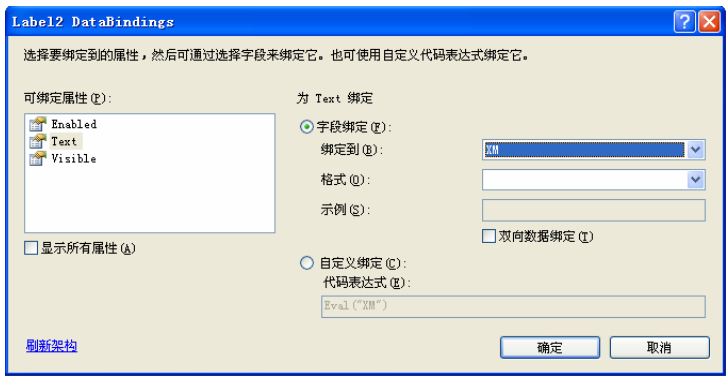


图 6.63　设定 Label 的绑定

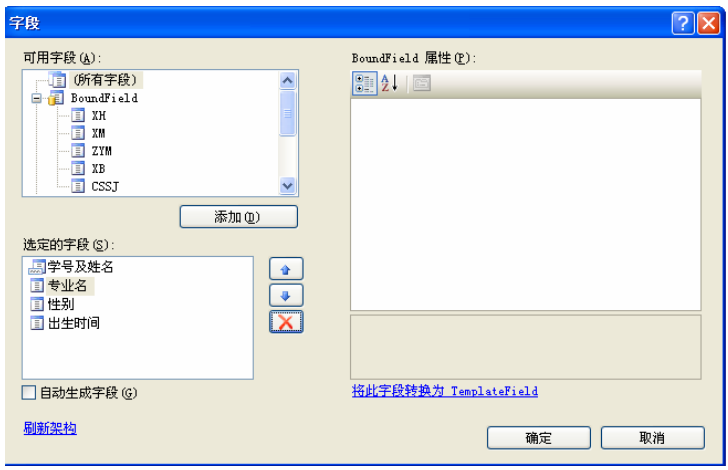


图 6.64　修改字段

运行程序，效果如图 6.65 所示，此时每个学生的学号和姓名都显示在同一列里面了。

图 6.65　运行结果

（3）使用 Calendar 控件显示“出生时间”字段

在 GridView 中将数据显示为文本的话，只需要简单地使用 BoundField 就可以了。然而，在某些特定的场合，数据需要展示为一个特殊的Web 控件，而不是一个简单的文本。例如，将学生的出生时间高亮显示在一个 Calendar 控件中将会使界面显得更有特色。这样的自定义的数据显示就可以用 TemplateField 来做。

要做到这一点，先将“出生时间”这个绑定列转换成一个模板列，具体操作不再赘述。正如第（2）步中看到的那样，这个操作会将绑定列替换成含有 ItemTemplate 和 EditItemTemplate 的模板列，它们各带有一个 Label 和 TextBox，而这个 Label 和 TextBox 的 Text 属性都使用了数据绑定语句<%# Bind("HiredDate")%>来将“出生时间”字段绑定到自身。在设计器中，从 GridView 的智能标签的弹出菜单中选择“编辑模板”（Edit Templates），并在下拉列表中选择“出生时间”模板列的 ItemTemplate。删除 Label 控件并从工具箱中拖一个 Calendar 控件到模板编辑界面中，如图 6.66 所示。

图 6.66　编辑模板列的 ItemTemplate

此时，GridView 中每一行的“出生时间”模板列都会包含一个 Calendar 控件，Calendar 控件默认显示当前日期。下面将学生的出生时间赋值给 Calendar 控件的 SelectedDate 和 VisibleDate 属性。从 Calendar 控件的智能标签中选择“编辑 DataBindings”，然后把 SelectedDate 和 VisibleDate 这两个属性都绑定到“出生时间”字段上，如图 6.67 所示。

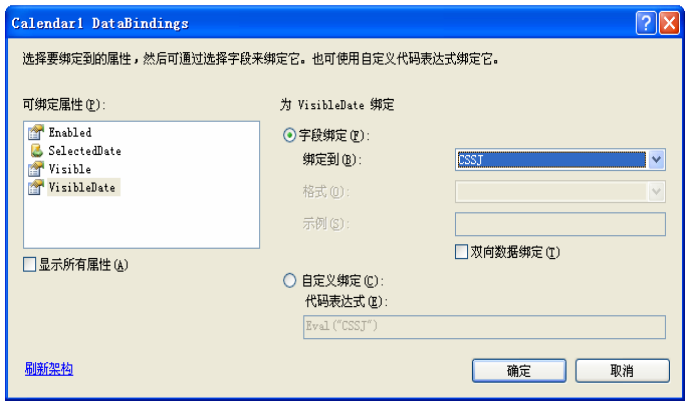


图 6.67　编辑 DataBindings

运行程序，可以看到Calendar 控件高亮显示的正是当前行的学生的出生时间，其效果如图 6.68 所示。

图 6.68　运行结果

（4）根据学生的出生时间显示学生年龄

到目前为止，我们已经看到了 TemplateField 的两个应用：

① 将两个数据合并到一个列中。

② 用 Web 控件来展示数据，而非简单的文本。

第三种 TemplateField 的用法是，在 GridView 中显示定制的数据。例如，除了显示学生的出生时间，还希望用一列来显示这个学生的年龄，而这一列是根据某些规则计算得来的。与此类似的是，某学生表中有一个性别字段，其中存储了 M 或 F 这样的字符，用于表示此学生的性别，而希望在页面上将其显示为“男”或“女”。

这两种用法都可以采用在 ASP.NET 页面的后置代码类中创建一个供模板调用的格式化

方法来做到。这样的格式化方法将在模板中调用，语法跟前面的数据绑定语法是一样的。格式化方法可以接收若干个参数，但是必须返回一个字符串。这个返回的字符串是一个用于插入到模板中的HTML。另外，注意必须将此格式化方法标记为 protected 或 public，否则模板无法访问到它。格式化方法 DisplayAge 的代码如下，把其加入到页面的代码中。

```
protected string DisplayAge (object birthDate)
{
    if (birthDate == null)
    {    return "UnKnown";    }
    else
    {
        TimeSpan ts = DateTime.Now.Subtract(DateTime.Parse(birthDate.ToString()));
        return Convert.ToString(ts.Days / 365)  + "岁" ;
    }
}
```

由于 birthDay 可能会含有空值，所以必须在进行计算之前首先保证其值不为空。如果 birthDay 值为空的话，直接返回“Unknown”；如果不为空则计算当前时间与 birthDay 之间所隔的年数，并把它作为一个字符串返回即可。

要使用这个方法，需要在 GridView 的 TemplateField 中使用数据绑定语法来调用它，首先通过编辑列为 GridView 添加一个新的模板列，如图 6.69 所示。

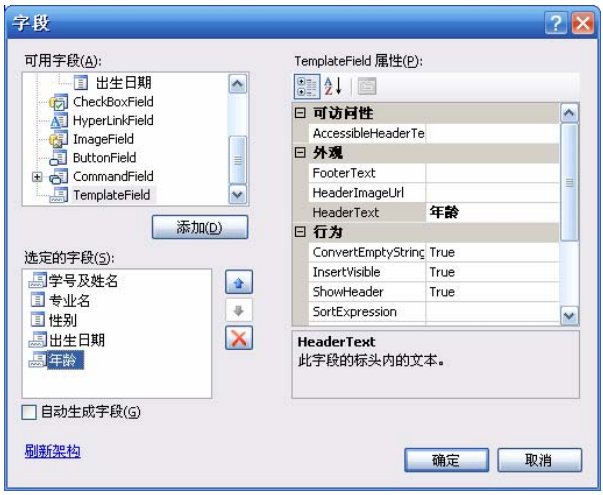


图 6.69　添加模板列

要调用DisplayAge 方法，需要在源代码视图给这个模板列添加一个ItemTemplate，并加上如下的数据绑定代码：

```
<ItemTemplate><%#DisplayAge(Eval("CSSJ")) %></ItemTemplate>
```

修改后的代码如下：

```
<asp:TemplateField HeaderText="年龄">
    <ItemTemplate><%#DisplayAge(Eval("CSSJ")) %></ItemTemplate>
</asp:TemplateField>
```

Eval("CSSJ ") 是普通的绑定表达式，用于获取“CSSJ”字段的值，作为 DisplayAge 方法的参数传递。运行程序，效果如图 6.70 所示。

图 6.70　运行效果

L4. FormView控件

FormView 控件可以呈现数据源中的单个记录，该控件与 DetailsView 控件相似。FormView 控件和 DetailsView 控件之间的差别在于，DetailsView 控件使用表格布局，在该布局中，记录的每个字段都各自显示为一行；而 FormView 控件不指定用于显示记录的预定义布局。实际上，可以创建一个包含控件的模板，以显示记录中的各个字段。该模板中包含用于创建窗体的格式、控件和绑定表达式。由于 FormView 不是由绑定列所组成，因此也就不能给一个 FormView 添加 BoundField 或 TemplateField。可以这样来理解 FormView，即把它当做只含有一个 TemplateField 的 DetailsView 控件。

FormView 控件依赖于数据源控件的功能执行如更新、插入和删除记录的任务。即使 FormView 控件的数据源公开了多条记录，该控件一次也只显示一条数据记录。

FormView 支持以下这些模板：

① ItemTemplate：用于在 FormView 中呈现一个特殊的记录。

② HeaderTemplate：用于指定一个可选的页眉行。

③ FooterTemplate：用于指定一个可选的页脚行。

④ EmptyDataTemplate：当绑定到 FormView 的数据源缺少记录的时候，EmptyDataTemplate 将会代替 ItemTemplate 来生成控件的标记语言。

⑤ PagerTemplate：如果 FormView 启用了分页的话，这个模板可以用于自定义分页的界面。

⑥ EditItemTemplate/InsertItemTemplate：如果 FormView 支持编辑或插入功能，那么这两种模板可以用于自定义相关的界面。

本节将学习如何使用 FormView 控件为学生信息呈现一个不规则的外观。还是使用上节中的“XSCJ”数据库中的“XS”表。FormView 的 ItemTemplate 将会使用一个页眉元素和<table>的结合体来显示姓名、学号、备注等的值，而不是使用各种各样的列。

（1）将数据绑定到 FormView

新建一个网站并添加一个命名为“Formview.aspx”页面，从工具箱中拖一个 FormView 到页面的“设计”视图中。与 GridView 和 DetailsView 不同的是，FormView 刚刚添加到页面上时，并没有一个预置的可视化界面，仅仅是一个灰色的方块，这就告诉我们它需要一个 ItemTemplate，而不是用简单的 BoundField 呈现外观，如图 6.71 所示。

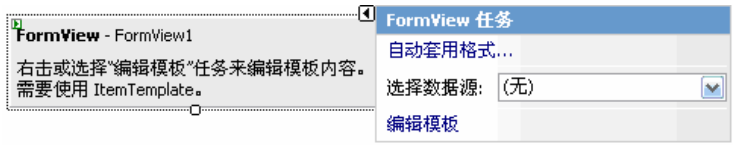


图 6.71　FormView 的设计视图

可以手工编写代码（在源视图中）为 FormView1 添加 ItemTemplate，也可以通过在“设计”视图中将 FormView1 绑定到一个数据源控件上来实现自动添加，这里将 FormView1 绑定到 SqlDataSource1 上（SqlDataSource1 按照上例进行设置，选择“XS”表中的除“ZP”字段外的所有字段）。绑定数据源的操作将为 FormView1 自动生成一个 ItemTemplate，在这个 ItemTemplate 中包含了用于显示各字段名称的 HTML 代码，还有用于显示各字段的值的 Label 控件，当然，这些 Label 控件的 Text 属性都已经绑定到了各相应的字段上。这个操作也同时生成了 InsertItemTemplate 和 EditItemTemplate，它们为数据源控件的每一个字段都呈现了一个输入控件。

切换到“源”视图，在标记代码中可以看到系统已经为绑定到数据源的FormView1 自动生成了 EditItemTemplate、InsertItemTemplate 和 ItemTemplate 模板。由于暂时不需要使用 FormView1 来编辑和插入记录，所以可以删除 EditItemTemplate 和 InsertItemTemplate。然后，清空 ItemTemplate 中的标记语言代码，这样才可以工作于一个干净的环境上。修改之后的 FormView1 的声明标记代码如下所示：

```
<asp:FormView ID="FormView1" runat="server" DataKeyNames="XH"DataSourceID="SqlDataSource1">
<ItemTemplate>
    …..
</ItemTemplate>
</asp:FormView>
```

接下来在 FormView1 的智能标签中勾上“启用分页”复选框，这样可以在 FormView 的声明标记代码中加上“AllowPaging="True"”属性。

（2）定义 ItemTemplate 的标记代码

在将 FormView 绑定到 SqlDataSource 控件并且将其配置为支持分页之后，就可以指定 ItemTemplate 的内容了。我们希望将学生姓名显示在<h3>标签中。紧跟着使用<table>将其他的学生信息显示在一个 3 行 4 列的表中，将学生信息属性名称放入第 1 列和第 3 列，并替换成对应的汉字，第 2 列和第 4 列用于显示学生信息属性的值。

在“设计”视图中通过 FormView 的模板编辑界面，或是在源视图中手工输入代码，都可以添加上面所说的这些标记代码。不论采用哪种方法，在定义好ItemTemplate 的内容后，FormView 的 ItemTemplate 模板外观如图 6.72 所示。

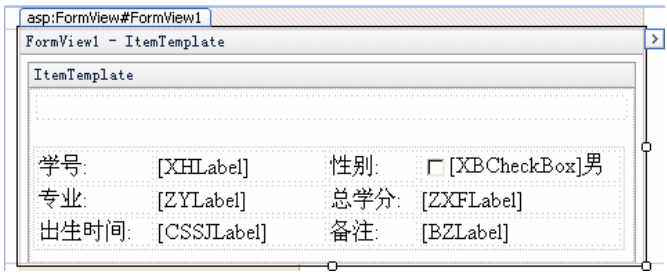


图 6.72　模板外观

注：标记代码中所用到的数据绑定语法，以<%#Eval("XM")%>为例，它们可以直接插入到模板的输出中，没有必要把字段值绑定到一个 Label 控件的 Text 属性上。例如，要将姓名的值使用<h3><%# Eval("XM")%> </h3>显示在一个<h3>元素中，那么学生“王林”将被输出为<h3>王林</h3>。

由于 FormView 没有 CheckBoxField，要将 XB（性别）的值显示为一个 CheckBox 的话，必须添加一个 CheckBox 控件。将这个 CheckBox 控件的 Enabled 属性设置为 False 以使其只读，并将其 Checked 属性绑定到“XB”字段上去即可。

完成了 ItemTemplate 之后，学生信息就以一种自己想要的方式来显示了。运行网页，其效果如图 6.73 所示。

图 6.73　运行结果

通过对 FormView 的学习，可以看到，虽然 GridView 和 DetailsView 控件可以使用 TemplateField 来自定义它们的输出，不过它们都呈现为方方正正的表格形式。在那些需要使用一种不规则的外观来显示一个单独的记录的时候，FormView 就是一个理想的选择。就像在本节中看到的那样，在显示一个单独的记录的时候，FormView 提供了更加复杂的呈现方式。

L5. DetailsView控件

DetailsView 控件类似于 FormView，只不过 FormView 提供了可以定制的展现模板，而 DetailsView 只能以表格的形式展现数据。

使用 DetailsView 控件可以从它的关联数据源中一次显示、编辑、插入或删除一条记录。默认情况下，DetailsView 控件将记录的每个字段显示在它自己的一行内。DetailsView

控件通常用于更新和插入新记录，并且通常在主/详方案中使用，在这些方案中，主控件的选中记录决定要在 DetailsView 控件中显示的记录。即使 DetailsView 控件的数据源公开了多条记录，该控件一次也仅显示一条数据记录。

DetailsView 控件依赖于数据源控件的功能执行如更新、插入和删除记录等任务，但是它不支持排序。

L6. ListView

Visual Studio 2008 提供了 ListView 和 DataPaper 这两个与数据有关的控件。ListView 是一个很强大的控件，可以实现其他数据控件实现的任意功能。而且ListView 也前所未有地灵活。通过定义它的模板，几乎可以实现任意一种数据展现方式。ListView 提供了默认的 5 种展现样式：Grid、Tiled、Bulleted List、Flow 和 SingleRow，也可以通过其模板进行自己的定制。

ListView 除了拥有数据展现功能外，还提供了数据维护的功能，可以在ListView 内部实现数据的插入、修改和删除等操作，无须再单独提供相应的界面。

第 7 章　设计成绩录入与打印模块

◇ 任务目标

本章主要目标是实现学生成绩的录入、打印和课程成绩信息查询功能，完成后的效果如图 7.1、图 7.2、图 7.3 所示。

图 7.1　录入学生成绩

图 7.2　学生成绩打印

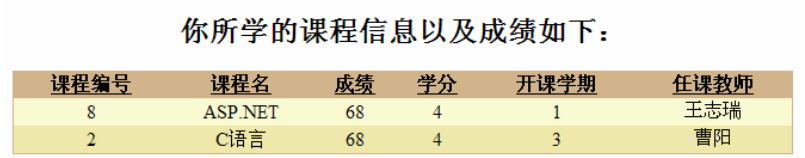


图 7.3　学生课程成绩信息显示页面

第一部分　应 用 实 践

7.1　学生成绩录入

教师登录系统后，进入成绩录入模块，首先列出未录入成绩的所有班级列表，在列表中单击某个班级就可以为该班级内的所有学生进行成绩的录入，录入结束后统一插入到数据库中。

1．规划学生成绩录入界面（见图 7.4）

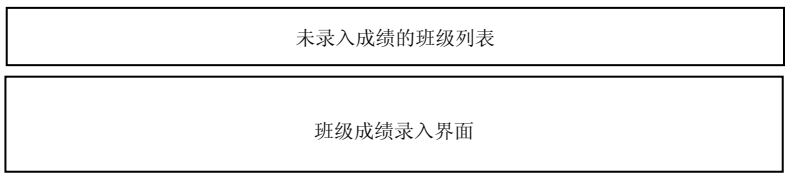


图 7.4　页面基本布局

2．设计学生成绩录入页面

（1）在前文所创建的网站项目“解决方案资源管理器”中打开 teacher 文件下的 enterScore.aspx 页面，切换到“源”视图，在 ID 为“Content2”的标记中添加一个 ID 为“content”的 DIV，设置其 Style 属性值为“border-style: solid; border-width: 1pt”。

（2）在 ID 为“content”的 DIV 内添加一个 DIV，设置其 ID 属性为“CourseList”。把光标定位到此层内，单击“应用样式”窗口中的“innerLayout”样式规则，给该层应用样式规则。

（3）把光标定位到 ID 为“CourseList”的 DIV 内，在光标位置插入一个 GridView 控件和 SqlDataSource 控件，设置 GridView 控件的 ID 为“GridViewCourseList”，设置 SqlDataSource 控件的 ID 为“SqlDataSourceCourseList”，选中“SqlDataSourceCourseList”控件，通过“属性”窗口设置其 ConnectionString 为“SqlServer2005ConnectionString”，设置 SelectQuery 为：

```
SELECT a.classID,c.className, a.courseID,c2.courseName FROM Arrange a
LEFT JOIN Class c ON c.classID = a.classID
LEFT JOIN Teacher t ON t.teaID = a.teaID
LEFT JOIN Course c2 ON c2.courseID = a.courseID
WHERE a.isRecord='0' AND a.teaID=@teaID
```

并设置@teaID 的参数源为“Session”，SessionFiled 为“teaID”。

（4）通过“GridView 任务”设置 GridViewCourseList 控件的数据源为“SqlDataSource-CourseList”。单击“刷新架构”选项，使 GridViewCourseList 自动绑定所查询出来的列，单击“编辑列”选项打开“字段”对话框，设置每个字段的“HeaderText”为中文描述，选中“可用字段”列表中的“ButtonFiled”，单击“添加”按钮加入到“选定的字段”中，并设置其 CommandName 为“InputScore”，Text 为“录入成绩”，设置后的效果如图 7.5 所示。

图 7.5　课程班级列表

（5）切换到“源”视图，在 ID 为“CourseList”的 DIV 下部插入一个 DIV，设置其 ID 为“StudentList”，通过“应用样式”为其应用“innerLayout”样式，并在层内插入一个 GridView 控件和一个 SqlDataSource 控件，设置 GridView 控件的 ID 为“GridViewStudentList”，设置 SqlDataSource 控件的 ID 为“SqlDataSourceStudentList”。

（6）切换到“设计”视图，通过“属性”窗口，设置 SqlDataSourceStudentList 控件的 ConnectionString 为“SqlServer2005ConnectionString”，设置 SelectQuery 为“SELECT s.stuID, s.stuName FROM Student s”，设置 GridViewStudentList 控件的数据源为“SqlDataSource-StudentList”。单击“GridView 任务”中的“刷新架构”选项，使 GridView-StudentList 自动绑定所查询出来的列，选中“启用分页”复选框，通过“编辑列”选项，打开“字段”对话框，设置每个字段的“HeaderText”为对应的中文描述，选中“可用字段”列表中的“TemplateField”，单击“添加”按钮加入到“选定的字段”中，设置 Header-Text 为“课程成绩”。

（7）单击“GridView 任务”中的“编辑模板”选项，进入“模板编辑”视图，在“ItemTemplate”内录入“考试成绩:”，并在其后加入一个 TextBox 控件，在“GridView 任务”中单击“结束模板编辑”选项结束模板编辑。在 SqlDataSourceStudentList 下部加入一个 Button 控件，设置 Text 为“完成录入”，设置 ID 为“ButtonInputScore”，设置后的效果图如图 7.6 所示。

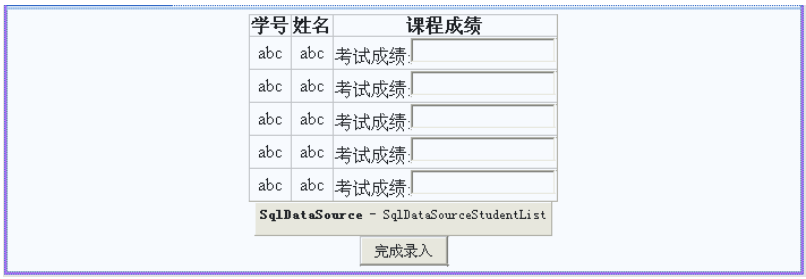


图 7.6　学生列表

3．编写学生成绩录入后台代码

（1）因为涉及操作 SQL Server 数据库，所以要添加如下命名空间：

```
using System.Data.SqlClient;
```

（2）页面初次显示的时候，只显示课程列表，学生列表部分先不显示，当单击“录入成绩”按钮后再显示所要录入的班级学生列表，因此在页面的 Load 事件内需添加如下的代码：

```
//页面初次显示的时候，隐藏成绩录入界面
if (!Page.IsPostBack)
{
    this.GridViewStudentList.Visible = false;
    this.ButtonInputScore.Visible = false;
}
```

（3）当单击 GridViewCourseList 控件内的“录入成绩”按钮时，自动显示对应班级的学生名单，因此需要在 GridViewCourseList 控件的 RowCommand 事件内添加如下的代码：

```
//如果单击的是“录入成绩”按钮，则执行如下的代码
if (e.CommandName == "InputScore")
{
    //获取所单击的行索引
    int index = Convert.ToInt32(e.CommandArgument);
    //获取班级编号和课程编号
    string ClassID = this.GridViewCourseList.Rows[index].Cells[0].Text.Trim();
    string CoureseID = this.GridViewCourseList.Rows[index].Cells[2].Text.Trim();
    //设置需要查询的班级
    string sql = "SELECT s.stuID, s.stuName FROM Student s WHERE s.stuClassID='" + ClassID + "'";
    this.SqlDataSourceStudentList.SelectCommand = sql;
    //显示成绩录入视图
    this.GridViewStudentList.Visible = true;
    this.ButtonInputScore.Visible = true;
    //保存当前所操作的班级和课程编号
    ViewState["ClassID"] = ClassID;
    ViewState["CoureseID"] = CoureseID;
}
```

（4）当成绩录入结束，单击“完成录入”按钮后，需对所录入的数据进行检测，合法后，把成绩信息添加到数据库中，并设置对应课程已录过成绩。因此在“完成录入”按钮的 Click 事件内添加如下的代码：

```
//对数据进行检查
for (int i = 0; i < this.GridViewStudentList.Rows.Count; i++)
{
    TextBox input = (TextBox)this.GridViewStudentList.Rows[i].Cells[2].FindControl("TextBox1");
```

```
        string inputText = input.Text.Trim();
        try
        {       int score = Convert.ToInt32(inputText);       }
        catch
        {
            //提示错误信息
            Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "wanging",
                                "alert('录入有误，请更正')",true);
        }
    }
    //通过事务把成绩插入到数据库中
    commDBHelper operation = new commDBHelper();
    //开始事务
    operation.BeginTransaction();
    List<SqlParameter> para = new List<SqlParameter>();
    try
    {
        for (int i = 0; i < this.GridViewStudentList.Rows.Count; i++)
        {
            TextBox input =(TextBox)this.GridViewStudentList.Rows[i].Cells[2].FindControl("TextBox1");
            string studentID = this.GridViewStudentList.Rows[i].Cells[0].Text.Trim();
            string inputText = input.Text.Trim();
            int score = Convert.ToInt32(inputText);
            para.Clear();
            string sql = "INSERT INTO Score(stuID,courseID,score)
                                        VALUES (@stuID,@courseID,@score)";
            para.Add(commDBHelper.CreateParameters("@stuD",DbType.String,studentID));
            para.Add(commDBHelper.CreateParameters("@courseId", DbType.Int32,
                                        (string)ViewState["CoureseID"]));
            para.Add(commDBHelper.CreateParameters("@score",DbType.Int32,score.ToString()));
            operation.ExecuteNonQueryWithTransaction(sql, CommandType.Text, para);
        }
        //修改该课程已录过课程成绩
        para.Clear();
        string sqlUpdate = "UPDATE Arrange SET isRecord = @isRecord
                        WHERE classID=@classID and courseID=@courseID and teaID=@teaID";
        para.Add(commDBHelper.CreateParameters("@isRecord",DbType.String,"1"));
        para.Add(commDBHelper.CreateParameters("@classID",DbType.String,
                        (string)ViewState["ClassID"]));
        para.Add(commDBHelper.CreateParameters("@courseID", DbType.Int32,
                         (string)ViewState["CoureseID"]));
```

```
        para.Add(commDBHelper.CreateParameters("@teaID",DbType.Int32, Session["teaID"].ToString()));
        operation.ExecuteNonQueryWithTransaction(sqlUpdate, CommandType.Text, para);
        //执行事务
        operation.Transtion.Commit();
        //提示操作成功
        Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "wanging", "alert('录入成功')",true);
        this.GridViewCourseList.DataBind();
        this.GridViewStudentList.Visible = false;
        this.ButtonInputScore.Visible = false;
    }
    catch
    {       operation.Transtion.Rollback();     }                   //事务回滚
    finally
    {       operation.EndTransaction();         }                   //事务结束
```

4．在浏览器中使用本页面

（1）在“解决方案资源管理器”中打开 Default.aspx 登录页面，按 Ctrl+F5 键运行此页面，登录一个已经注册并安排了课程的教师，将出现如图 7.7 所示界面，显示出当前登录教工所有需要录入成绩的课程列表。

图 7.7　学生列表

（2）单击“录入成绩”按钮，出现如图 7.8 所示的界面，让教师给对应班级录入课程成绩。

图 7.8　录入学生成绩页面

（3）在文本框内录入成绩后，单击“完成录入”按钮，如果录入的数据不合法，将提示错误；如果合法，则提示“录入成功”提示框。录入结束后，该班级就会自动隐藏掉，不能再次录入成绩。

7.2　学生成绩查询与打印

学生成绩查询与打印模块用来实现根据学生的班级和课程查询出本班所有学生的课程成绩，并且能够以班级为单位实现成绩打印功能，学生成绩查询页面布局如图 7.9 所示。

1．规划学生成绩查询与打印界面

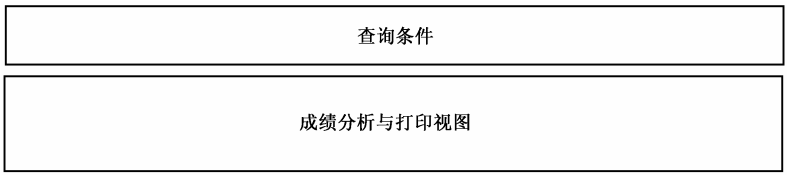


图 7.9　学生成绩查询页面布局

2．设计学生成绩查询与打印界面

（1）打开前文所创建的网站项目，右击“解决方案资源管理器”内的 Admin 文件夹，选择“添加新项”选项，在弹出的对话框中选择“Web 窗体”，名称修改为“SearchScore.aspx”，语言选择“Visual C#”，选中“选择母版页”，完成后单击“添加”按钮，在出现的窗口中选择 admin 文件夹下的 adminMasterPage.master，单击“确定”按钮完成新页面的创建。

（2）打开 Web.sitemap 文件，在<siteMapNode url="" title="管理员子系统" description="" >标记内部加入如下内容：<siteMapNode url="~/admin/SearchScore.aspx" title="学生成绩查询分析与打印" description="" />。

（3）切换到 SearchScore.aspx 页面的“设计”视图，在空白的位置右击，选择“属性”，在“属性”窗口中的下拉框中选择“Document”文档对象，设置 StyleSheetTheme 属性为“主题 1”。

（4）打开页面的“源”视图，把光标定位到 ID 为“Content2”的标记内，添加一个 ID 为“content”的 DIV，设置 Style 属性值为“border-style: solid; border-width: 1pt”。

（5）在 ID 为“content”的 DIV 中添加一个 DIV，并设置其 ID 属性为“Search”。把光标定位到此层内，单击“应用样式”窗口中的“innerLayout”样式规则，给该层应用样式规则。

（6）切换到“设计”视图，把光标定位到“Search”层内，按照如图 7.10 所示添加相应的控件，设置第一个 DropDownList 控件的 ID 为“DropDownListClass”，并通过 Items 属性添加一个项，项的 Text 属性为“选择班级”，项的 Value 属性为“null”，AppendDataBoundItems 属性为“True”；设置第二个 DropDownList 控件的 ID 属性为“DropDownListCourse”，并通过 Items 属性添加一个项，项的 Text 属性为“选择课程”，项的 Value 属性为“null”，AppendDataBoundItems 属性为“True”；Button 控件的 ID 为“ButtonSearch”，Width 为“80 px”，Text 设置为“搜索”。

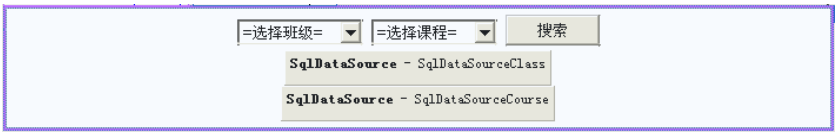


图 7.10　学生成绩查询界面布局

（7）设置第一个 SqlDataSource 控件的 ID 为“SqlDataSourceClass”，Connection-String 属性为“SqlServer2005ConnectionString”，SelectQuery 属性为“SELECT classID, className FROM Class”。设置第二个 SqlDataSource 控件的 ID 为“SqlDataSourceCourse”，ConnectionString 属性为“SqlServer2005ConnectionString”，SelectQuery 属性为“SELECT courseID, courseName FROM Course”。设置 DropDownListClass 的数据源为“SqlDataSourceClass”，设置在 DropDownList 中显示的数据字段为“className”，设置 DropDowinList 的值选择字段为“classID”。设置 DropDownListCourse 的数据源为“SqlDataSourceCourse”，设置在 DropDownList 中显示的数据字段为“courseName”，设置 DropDowinList 的值选择字段为“courseID”。

（8）在“解决方案资源管理器”内，右击站点，选择“添加新项”选项，在出现的窗口中选择“数据集”模板，修改文件的名称为“ScoreDataSet.xsd”，单击“添加”按钮，将提示是否把文件保存到 App_Code 文件夹，单击“是”按钮。

（9）在出现的“数据集设计器”界面内，选择“添加-TableAdapter”，弹出“TableAdapter 配置向导”，选择连接串为“SqlServer2005ConnectionSring”，单击“下一步”按钮，在弹出的“TableAdapter 应如何访问数据库”对话框中，选择“使用 SQL 语句”，单击“下一步”按钮，在“输入 SQL 语句”对话框中添加如下的语句：

```
SELECT
        s.stuID,s2.stuName,c2.className,
        s.courseID,c.courseName,
        s.score
FROM
        Score s
        LEFT JOIN Course c ON c.courseID = s.courseID
        LEFT JOIN Student s2 ON s2.stuID = s.stuID
        LEFT JOIN Class c2 ON c2.classID = s2.stuClassID
```

单击“完成”按钮完成创建工作，创建结束后的界面如图 7.11 所示。

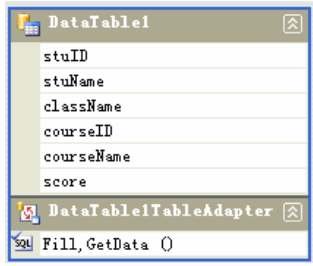


图 7.11　DataTable

（10）在“解决方案资源管理器”内，右击 Admin 文件夹，选择“添加新项”选项，在弹出的“添加新项”对话框中选择“**Crystal 报表**”模板，单击“添加”按钮后弹出“Crystal Reports 库”对话框，选中“作为空白报表”，单击“确定”按钮完成创建，同时打开所创建的 CrystalReport.rpt 文件。选择“Crystal Reports”→“设计”→“打印机设置”菜单项，在弹出的“打印设置”对话框中设置纸张大小为“A5”。

（11）选择“Crystal Reports”→“数据库”→“数据库专家”菜单项，打开“数据库专家”对话框，依次展开“项目数据”→“ADO.NET 数据集”→“ScoreDataSet”菜单项，选中“DataTable1”，单击 > 图标，把其加入到“选定的表”内，单击“确定”换钮完成数据源的添加。

（12）从“工具箱”内拖放一个文本对象到报表头中，并在内部录入“学生成绩单”，选中并右击“学生成绩单”，选择“文本格式”选项，在弹出的“文本格式”对话框中设置字体的格式为“20 pt”，调整文本框到合适的大小。打开“字段资源管理器”窗口，展开“数据库字段”内的“DataTable1”字段，如图 7.12 所示，把对应的字段拖放到“详细资料”栏中，并设置“页眉”栏内的文字描述内容。从“工具箱”内拖放一个线条对象到报表内，在线条对象的属性窗口中设置 LineThickness 为“40”，设置后如图 7.12 所示。

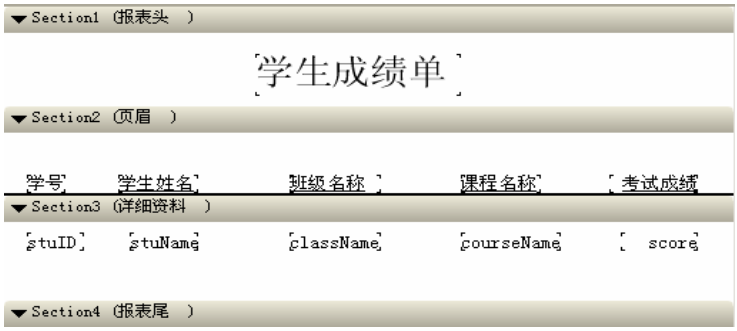


图 7.12　水晶报表设计

（13）打开 SearchScore.aspx 的“源”视图，在 ID 为“Search”的 DIV 下部插入一个 DIV，设置其 ID 为“Result”，通过“应用样式”为其应用“innerLayout”样式，并在此层内插入一个“**CrystalReportViewer”控件**。

3．编写学生成绩查询与打印后台代码

（1）因为要对报表进行操作，所以要添加如下的命名空间。

```
using CrystalDecisions.CrystalReports.Engine;
```

（2）当用户单击“搜索按钮”的时候，根据用户的选择进行搜索，并把所有的结果通过水晶报表显示出来，因此需要在 ButtonSearch 的 Click 事件内添加如下的代码：

```
//如果课程有选择，并且班级也有选择
if (this.DropDownListClass.SelectedValue != "null"
        && this.DropDownListCourse.SelectedValue != "null")
{
    //分别获取课程和班级编号
    string classID = this.DropDownListClass.SelectedItem.Value.Trim();
```

```
        string CoureseID = this.DropDownListCourse.SelectedItem.Value.Trim();
        //根据选择定义查询语句
        string sql = @"SELECT
                                s.stuID,s2.stuName,c2.className,
                                s.courseID,c.courseName,
                                s.score
                        FROM Score s
                                LEFT JOIN Course c ON c.courseID = s.courseID
                                LEFT JOIN Student s2 ON s2.stuID = s.stuID
                                LEFT JOIN Class c2 ON c2.classID = s2.stuClassID
                                WHERE c2.classID='" + classID + "' and s.courseID='" + CoureseID +
                                        "' order by s.stuID DESC";
        //把返回结果保存到 DataTable 中
        DataTable table = commDBHelper.ExecuteReader(sql, CommandType.Text, null);
        //如果查询有结果
        if (table.Rows.Count > 0)
        {
            ViewState["result"] = table;
            //创建报表对象，并设置需要加载的报表文件
            ReportDocument doc = new ReportDocument();
            doc.Load(Server.MapPath(@"~/Admin/CrystalReport.rpt"));
            doc.SetDataSource(table);
            this.CrystalReportViewer1.Visible = true;
            this.CrystalReportViewer1.ReportSource = doc;
            //对 Viewer 外观进行设置
            CrystalReportViewer1.HasCrystalLogo = false;
            CrystalReportViewer1.HasSearchButton = false;
            CrystalReportViewer1.HasToggleGroupTreeButton = false;
            CrystalReportViewer1.DisplayGroupTree = false;
        }
        else
        {
            //报错信息
            this.CrystalReportViewer1.Visible = false;
            Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "wanging",
                        "alert('成绩还没录入')", true);
        }
    }
    else
    {
```

```
        //报错信息
        Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "wanging",
                                "alert('请选择课程和班级')", true);
        this.CrystalReportViewer1.Visible = false;
    }
```

（3）为了在页面回送以后还能继续显示报表，需要在页面的 Load 事件内添加如下的代码：

```
    //如果已经显示过，则重新进行加载
    if (ViewState["result"] != null)
    {
        DataTable table = (DataTable)ViewState["result"];
        //创建报表对象，并设置需要加载的报表文件
        ReportDocument doc = new ReportDocument();
        doc.Load(Server.MapPath(@"~/Admin/CrystalReport.rpt"));
        doc.SetDataSource(table);
        this.CrystalReportViewer1.Visible = true;
        this.CrystalReportViewer1.ReportSource = doc;
        //对 Viewer 外观进行设置
        CrystalReportViewer1.HasCrystalLogo = false;
        CrystalReportViewer1.HasSearchButton = false;
        CrystalReportViewer1.HasToggleGroupTreeButton = false;
        CrystalReportViewer1.DisplayGroupTree = false;
    }
```

4．在浏览器中使用本页

（1）在“设计”视图，右击选择“在浏览器中查看”，出现如图 7.13 所示界面。

图 7.13　查询页面

（2）分别选择一个班级和一门课程，单击“搜索”按钮，显示如图 7.14 所示的界面。

（3）单击工具条上的 和 图标，可以实现成绩单的导出和打印功能。单击“打印”按钮，则会弹出一个窗口让选择打印的范围，单击“确定”按钮，即可进行打印。单击“导出”按钮，则会弹出一个窗口让选择导出的格式和范围，设置完后，单击“确定”按钮可以把成绩单导出为别的格式，方便进行编辑处理。

图 7.14　学生成绩报表

5．完善菜单功能

至此，整个学生成绩管理系统设计基本完成，但在Admin 和 Teacher 文件夹中的 adminMasterPage.master 和 teacherMasterPage.master 母版页中所添加的菜单控件还没设计完成，需要添加菜单链接到各个页面功能。

设计步骤如下。

因为各个母版页中的菜单控件都设置了“退出系统”功能，所以要设计一个页面提供退出系统功能，当退出系统时链接到此页面即可。

（1）添加退出页面（Quit.aspx）。在“解决方案资源管理器”中右击站点，选择“添加新项”选项，在弹出的“添加新项”对话框中选择“Web 窗体”模板，并命名为“Quit.aspx”，单击“添加”按钮在站点根目录下添加一个网页。

（2）添加退出系统代码。打开 Quit.aspx.cs 代码页，在 Page_Load 方法中添加如下代码：

```
Response.Write("<script>window.opener=null;window.close();</script>");          //关闭系统
Response.Write("<script language=javascript>history.go(-1);</script>");          //返回到前面页
```

当单击“退出系统”链接到此页面时，执行第一条语句并弹出对话框提示是否关闭系统，若选择“是”则关闭系统；若选择“否”，则执行第二条语句返回到前面页面。

（3）设置管理员母版页中菜单控件。打开 adminMasterPage.master 母版页中菜单控件属性窗口，选中“Items”，单击后面出现的  图标，在弹出的“菜单项编辑器”对话框中设置“教工维护”的 NavigateUrl 为“~/Admin/teacher.aspx”、“课程维护”的 NavigateUrl 为“~/Admin/course.aspx”、“排课维护”的 NavigateUrl 为“~/Admin/ArrangeCourse.aspx”、“打印成绩”的 NavigateUrl 为“~/Admin/SearchScore.aspx”、“退出系统”的 NavigateUrl 为“~/Quit.aspx”。

（4）设置教师母版页中菜单控件。与管理员母版页中的菜单控件设置一样，其中“录入学生成绩”的 NavigateUrl 为“~/Teacher/enterScore.aspx”、“打印学生成绩单”的 NavigateUrl 为“~/Admin/SearchScore.aspx”、“退出系统”的 NavigateUrl 为“~/Quit.aspx”。

7.3　学生成绩信息显示

学生登录系统后，进入课程成绩信息显示模块，显示登录学生的所学课程成绩和课程信息。页面布局如图 7.15 所示。

1．规划学生成绩信息显示页面

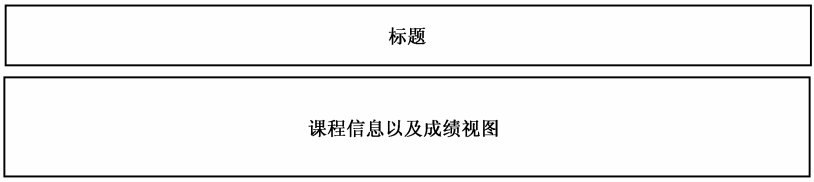


图 7.15　学生成绩信息显示界面布局

2．设计学生成绩信息显示页面

（1）添加新页面。打开前文所创建的网站项目，右击“解决方案资源管理器”内的 Students 文件夹，选择“添加新项”选项，在弹出的对话框中，选择“Web 窗体”，名称修改为“StuSearchScore.aspx”，语言选择“Visual C#”，不选中“选择母版页”复选框，单击“添加”按钮完成新页面的创建。

（2）添加页面标题。打开 StuSearchScore.aspx 页面的“源”视图，在“<body>”下添加语句“<h2>你所学的课程信息以及成绩如下：</h2>”。

（3）添加控件。在“源”视图中给标签“ ”添加 ID 属性，属性值为“center”。打开属性窗口，在顶部的下拉框中选择“Document”，设置 StyleSheetTheme 属性为“主题 1”。从工具箱中拖放一个 GridView 控件和 SqlDataSource 控件到 标签中。

（4）设置控件。切换到“设计”视图。打开“SqlDataSource1”的属性窗口，ConnectionString 设置为“SqlServer2005ConnectionString”。选中 SelectQuery 属性，单击 图标，弹出“命令和参数编辑器”对话框，添加 SELECT 命令如下：

```
SELECT c.courseID,c.CourseName,s.score,c.courseScore ,a.team ,t.teaName FROM
    Arrange a
LEFT JOIN Course c ON c.courseID=a.courseID
LEFT JOIN Score s ON s.courseID=a.courseID
LEFT JOIN Teacher t ON t.teaID=a.teaID
WHERE (s.stuID = @stuID)
```

单击“刷新参数”按钮，“参数源”选择“Session”，“SessionField”中输入“userID”，单击“确定”按钮完成命令编辑。选择“GridView1”，单击 图标，在“GridView 任务”中设置数据源为“SqlDataSource1”，选中“分页”和“排序”两个复选框，单击“刷新架构”选项，弹出“刷新数据源架构”对话框，单击“确定”按钮后 GridView1 重新刷新了架构。在“GridView 任务”中单击“编辑列”选项，在弹出的“字段”对话框中将各个字段的 HeaderText 属性值改为对应的字样，分别为“课程编号”、“课程名”、“成绩”、“学分”、“开课学期”和“任课教师”，单击“确定”按钮完成字段的编辑。

在“GridView 任务”中选择“自动套用格式”选项，在弹出的“自动套用格式”对话框中选择“沙滩和天空”格式，单击“确定”按钮完成格式的套用。

3．在浏览器中显示本页

（1）打开登录页面 Default.aspx，按 Ctr+F5 键运行页面，在“用户名”处输入学生学号，输入密码和选择学生登录后，单击“登录”按钮链接到 StuSearchScore.aspx 页面，显示此学生的所学的课程成绩及课程信息，如图 7.16 所示。

你所学的课程信息以及成绩如下：

课程编号	课程名	成绩	学分	开课学期	任课教师
8	ASP.NET	68	4	1	王志瑞
2	C语言	68	4	3	曹阳

图 7.16　学生课程成绩信息显示页面

4．设置起始页

至此，整个系统设计完成，下面设置此系统的起始页。在“解决方案资源管理器”中右击 Default.aspx 页面，选择“设为起始页”选项，则此页面设置为起始页面，即每次打开网站时首先运行的页面。

第二部分　知识点链接

L7.1　学生成绩查询与打印

L1. Crystal报表

水晶报表（Crystal Reports）是 Crystal Reports 公司开发的一个报表开发工具，自 1993 年开始就已经是 Visual Studio 的一部分，现在已经成为 Visual Studio 2008 中的标准报表创建工具。每套 Visual Studio 2008 都附带了该工具，并且被直接集成到开发环境中。

Crystal Reports for Visual Studio 2008 是内置于 Visual Studio2008 的报表设计工具，利用 Crystal Reports for Visual Studio 2008 能够在 Windows 环境中创建达到演示质量的交互式内容。使用 Crystal Reports for Visual Studio 2008 可在基于 GUI 和 Web 的程序中创建复杂、专业的报表。然后，可以将报表连接到几乎所有数据源及代理数据，如结果集（如一个 ADO.NET DataSet）。使用 GUI 设计器中附带的向导，可以方便地设置格式化、分组、图表制作和其他条件。

Crystal Reports for Visual Studio 2008 是以 Crystal Report 10.5 结构为基础，并针对.NET 平台做更进一步的强化与发展，以确保能提供给.NET 开发人员最丰富、完整的报表功能。

基本上，Crystal Reports for Visual Studio 2008 拥有如下的特色与优势：

① 针对所有以.NET 为目标的程序语言提供高度的集成，这意味着，可以在 Visual Basic .NET、Visual C#或 Visual C++的项目中使用本身所专长的程序语言来设计报表。

② 提供强大的报表设计工具 Crystal Report Designer。如图 7.17 所示，Crystal Report Designer 是所见即所得的报表设计工具，它可以自定义报表的数据源，让设计者在屏幕画面上定位各个控件（包括文本、线条、对话框、文本框、图片等），并且可轻易制作出各种复杂格式的报表（包括数据分组、数据运算、图标等）。

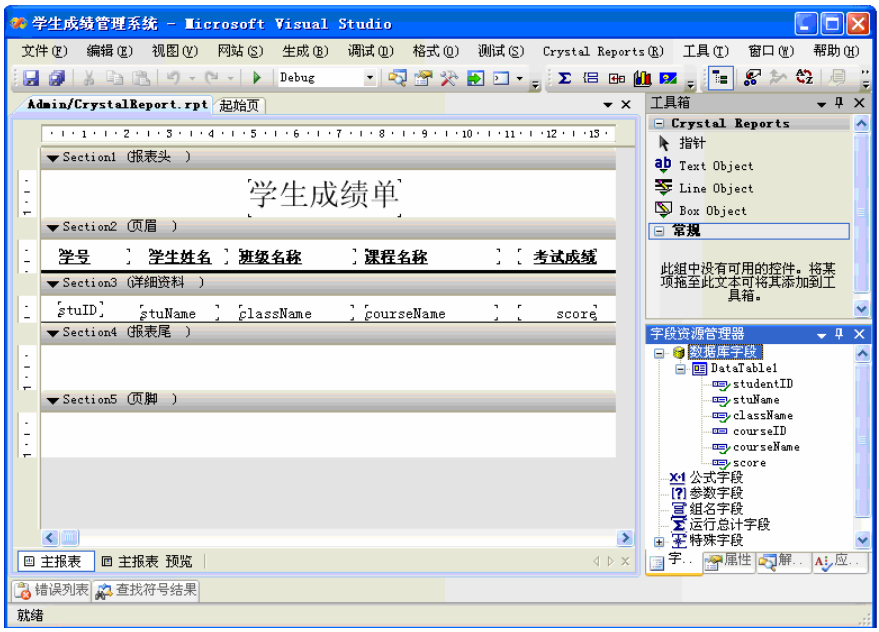


图 7.17　水晶报表设计器

③ 通过 Crystal Report for Visual Studio 2008 所提供的报表查看器控件（CrystalReportViewer），可以轻易将 Crystal Report Designer 所制作的报表展现在 Web Form 页面和 Windows Form 窗体上。

④ 由于 Web 应用程序所有的报表与报表 Web 服务处理都是在服务器上进行的，因此用户的电脑只需安装浏览器就能够访问 Web 应用程序的报表与享受报表 Web 服务，不需要安装额外的软件或组件。

⑤ 通用的标准文件格式支持功能丰富的报表，报表中可以包含数据检索条件、分组、摘要、参数、深化和子报表链接信息。

⑥ 数据和 .NET DataSet 连接，使用数据库专家可以方便地与多种数据库协议及 ADO.NET DataSet 形式的代理数据交互。

⑦ 使用导出功能可以将数据从 CrystalReportViewer 控件导出为 Word、Excel、PDF、HTML 和 Crystal Reports 格式。

⑧ 利用从 CrystalReportViewer 控件打印的功能，可在任何 Web 或 Windows 应用程序中实现基于页的报表打印。

⑨ 多语种客户端支持对 CrystalReportViewer 控件进行配置，以便在工具提示中显示其他语言，具体将视客户端浏览器、ASPX 页面或计算机的环境设置而定。

⑩ 在 ASP.NET Web 服务项目中，可以使用报表 Web 服务创建项目、添加 Crystal 报表，并将其作为 Web 服务发布。

⑪ 使用 Crystal Reports SDK 可以通过编程方式与报表进行交互，并且可以修改报表。可以使用 4 种不同的对象模型之一，每种的复杂度和功能逐渐增加。

L2. CrystalReportViewer控件

在应用程序中，通常都需要显示报表。在.NET 中，大多数情况使用了水晶报表，如果不是直接将报表发送到打印机打印，那么就需要将报表显示出来，这种情况下需要使用报表查看器（CrystalReportViewer）。

报表查看器（CrystalReportViewer）是 ASP.NET 中的服务器控件，主要用来实现把报表文档显示在 Web 或 Windows 应用程序中，并提供方便的报表打印功能（能够提供预览、报表放大和缩小等方便的功能）。报表查看器除了可以打印报表之外，还提供了报表数据到各种文档之间的转换功能，能够方便地把报表内的数据和图表转换为文档格式（Word、Excel、PDF、HTML 等）。

第 8 章　系统发布与部署

8.1　站点发布

软件开发完成后，往往需要把产品交付给客户，但是为了保护软件的知识产权，不可能直接把整个网站的源代码复制给客户，需要采用 Visual Studio 2008 所提供的站点生成功能对网站进行生成，生成以后，.aspx.cs 文件内的后台代码就会被编译为.dll 文件，最后把.aspx 文件和.dll 文件复制给客户即可。

可以通过如下的方式来发布站点：运行 Visual Studio 2008，打开系统站点，打开“解决方案资源管理器”窗口，在“解决方案资源管理器”内右击站点目录，选择“发布网站”，出现如图 8.1 所示的窗口。

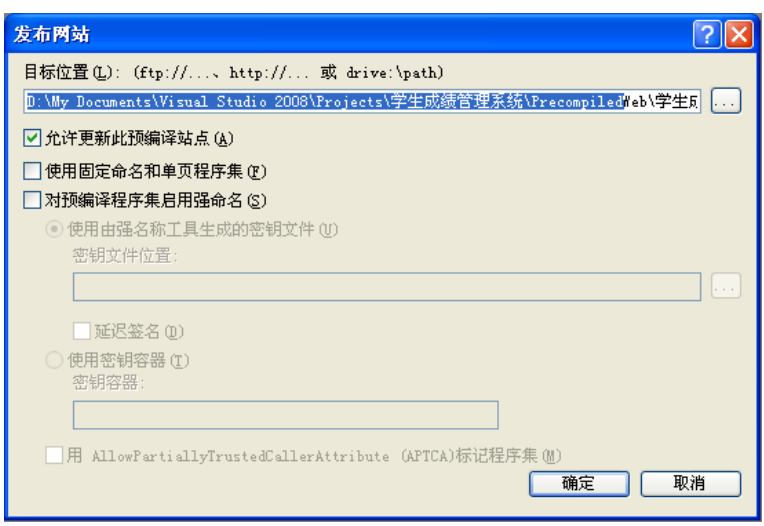


图 8.1　发布站点

目标位置：用来设置发布文件的存放路径。

允许更新此预编译站点：这是 Visual Studio 2008 默认的编译模式，在“发布网站”中选中“允许更新此预编译站点”，就是指定.aspx 页面的内容不编译到程序集中，而是将标记保留原样，从而能够在预编译网站后更改 .aspx 内容（方便美工人员对页面的.aspx 内容进行后期修改）。如果不选中此选项，则.aspx 页面内的内容也会被编译进程序集中，无法更改，页面内部只剩下如下的内容“这是预编译工具生成的标记文件，不应被删除!”。这种部署方式最大的问题在于，aspx 页面会在发布时自动指向程序集，而每次编译的程序集文件

名会发生变化，这就导致如果修改了任何一个程序逻辑并重新编译，就不得不上传所有网站内的程序文件。

使用固定命名和单页程序集：这个选项会解决上面的随机程序集命名问题，这个模式下的编译会把每个 ASPX 文件都编译为一个单程序集，每个程序集后会添加随机数，不过这个随机数是固定的。

对预编译程序集启用强命名：指定使用密钥文件或密钥容器，使生成的程序集具有强名称，以对程序集进行编码并确保它们未被恶意窜改，更加保证发布代码的安全性。

8.2 站点部署

ASP.NET 站点在开发过程中，可以使用 Visual Studio 2008 开发环境内置的虚拟服务器对程序进行测试，但是在实际的环境中，需要把其部署到操作系统内部的 IIS 服务器内才可以运行。除了需要在所部署的机器内部安装 IIS 外，还需要安装所对应版本的.Net Framework 框架，具体的安装与部署过程如下（以 Windows XP 为例）。

1．安装.NET Framework 框架

如果客户的电脑内没有安装.NET Framework 框架，需要到微软的官方网站下载对应版本的框架安装包进行安装，具体安装方法类似于普通的安装程序，这里就不具体介绍。

2．安装IIS服务器

（1）打开“控制面板”，双击“添加或删除程序”，弹出“添加或删除程序”对话框，在此对话框中单击“添加/删除 Windows 组件”，弹出如图 8.2 所示的“Windows 组件向导”对话框。

图 8.2 Windows 组件向导

（2）在光驱内插入系统的安装盘，在“组件”内选择“Internet 信息服务(IIS)”，单击“下一步”按钮开始组建的安装，成功安装后将弹出成功安装界面。

3．把应用程序部署到IIS内

（1）在 C 盘下创建一个 WebSite 文件夹，把发布好的站点复制到 WebSite 文件夹内。

（2）右击“我的电脑”，选择“管理”，打开“计算机管理”窗口，展开“服务和应用程序”的“Internet 信息服务”内的“网站”。

（3）右击“默认网站”，选择“属性”，打开如图 8.3 所示的“默认网站属性”窗口。

此窗口中有很多的选项卡，其中比较重要有“ASP.NET”、“网站”、“主目录”、“文档”，需要进行设置。

图 8.3　默认网站属性

ASP.NET 选项卡：主要用来设置 IIS 所使用的.NET Framework 版本，一般情况下需要先安装 IIS，再安装.NET Framework，这样系统可以正常解析.aspx 页面。但是，如果是先安装了.NET Framework，再安装 IIS，则会由于相关的 IIS 组件没有得到.NET Framework 的更新，导致无法正常解析.aspx 页面。解决方法如下：

① 运行 C:\Windows\Microsoft.NET\Framework\.v2.0.50727 目录下的 aspnet_regiis.exe 可执行文件，更新相关组件。

② 执行 DOS 命令“aspnet_regiis –r”，更新相关组件。

注：运行.NET 3.5 网站，在 IIS 管理器中的 ASP.NET 版本中只需选择.NET 2.0 就可以。因为 3.0、3.5 中都有额外的功能，但它们编译后的代码还是 2.0，.NET 框架是基于静态编译的代码，因此用 2.0 即可解释了。

网站选项卡：主要用来设置 IIS 网站所绑定的 IP 地址和服务端口，如图 8.4 所示。

主目录选项卡：主要用来设置站点的本地路径，并设置目录的访问权限，如图 8.5 所示。

文档选项卡：主要用来设置站点内默认的网站首页，如图 8.6 所示。

（4）设置完毕后，右击“默认网站”，选择“浏览”，查看是否能够显示出站点内的默认首页，如果能够显示则配置成功。

图 8.4　网站选项卡

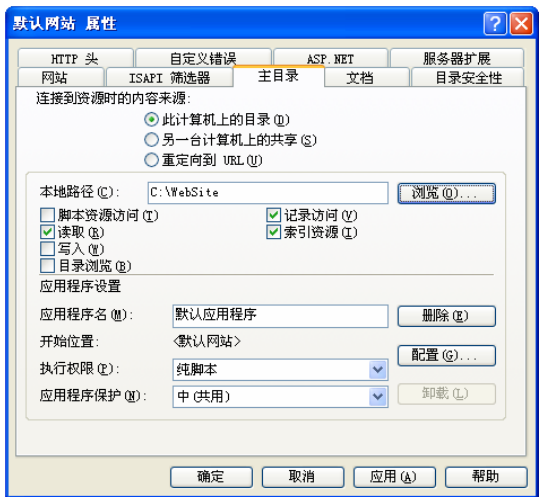


图 8.5　主目录选项卡

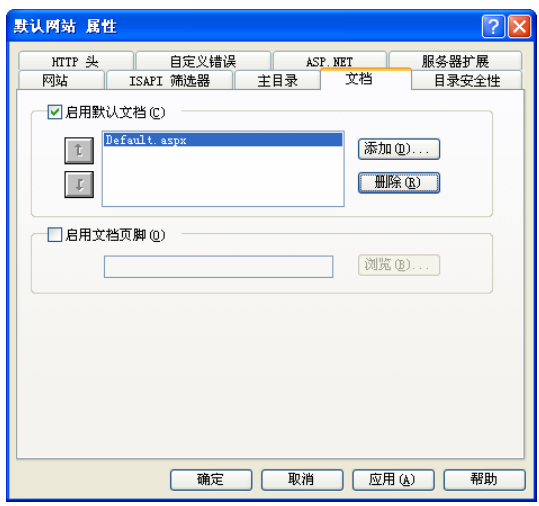


图 8.6　文档选项卡

附录A　C#常用语法简介

本附录旨在为读者学习 C#语言提供参考，帮助读者对 C#有一个初步的了解。附录中仅对 C#的基本语法做一介绍，未涉及 C#的所有特性和功能。事实上，C#是微软公司专为 .NET 平台量身打造的一种系统性的程序设计语言，并不是本附录介绍的基本语法内容所能完全涵盖的，如果需要了解更为详细的语言特性和语法细节，请参阅微软公司的 MSDN 帮助文档。

A.1　C# 简介

作为编程语言，C#是现代的、简单的、完全面向对象的，而且是类型安全的。重要的是，C#是一种现代编程语言。在类、名字空间、方法重载和异常处理等方面，C#去掉了C++中的许多复杂性，借鉴和修改了 Java 的许多特性，使其变得更易于使用，不易出错。

下面列举了一些 C#在设计上的优点。

（1）简单性

C# 在设计上的首要目标就是简单性。

没有指针是 C#的一个显著特性。在默认情况下，你使用一种可操控的（Managed）代码进行工作，此时一些不安全的操作（如直接的内存存取）将是不允许的。

在 C# 中不再需要记住那些源于不同处理器结构的数据类型，如可变长的整数类型。C#在 CLR 层面统一了数据类型，使得.NET 上的不同语言具有相同的类型系统，可以将每种类型看做一个对象，不管它是初始数据类型还是完全的类。

整型和布尔数据类型是完全不同的类型。这意味着 if 判别式的结果只能是布尔数据类型，如果是别的类型编译器将会报错。那种搞混了比较和赋值运算的错误将不会再发生了。

（2）现代性

许多在传统语言中必须由自己来实现或者干脆没有的特征，都成为基础 C#实现的一部分。金融类型对于企业级编程语言来说是很受欢迎的一个附加类型，可以使用一个新的 decimal 数据类型进行货币计算。

安全性是现代应用的头等要求，C#通过代码访问安全机制来保证安全性，根据代码的身份来源，可以分为不同的安全级别，不同级别的代码在被调用时将受到不同的限制。

（3）面向对象

C# 支持面向对象的所有关键概念：封装、继承和多态性。整个 C# 的类模型是建立在.NET 虚拟对象系统（VOS，Virtual Object System）之上的，而这个对象模型是基础架构的一部分，而不再是编程语言的一部分——它们是跨语言的。

C# 中没有全局函数、变量或常数。每样东西必须被封装在一个类中，或者作为一个实

例成员（通过类的一个实例对象来访问），或者作为一个静态成员（通过类型来访问），这会使 C#代码具有更好的可读性，并减少了发生命名冲突的可能性。

多重继承的优劣一直是面向对象领域争论的话题之一，然而在实际的开发中很少用到，在多数情况下从多个基类派生所带来的问题比这种做法所能解决的问题更多，因此 C#的继承机制只允许一个基类。如果需要多重继承，可以使用接口。

（4）类型安全性

当在 C/C++中定义一个指针后，可以自由地把它指向任意一个类型，包括做一些相当危险的事，如将一个整型指针指向双精度型数据。只要内存支持这一操作，它就会凑合着工作，这当然不是你所设想的企业级编程语言类型安全性。与此相反，C#实施了最严格的类型安全来保护它自身及其垃圾收集器。因此，程序员必须遵守关于变量的一些规定，如不能使用未初始化变量。对于对象的成员变量，编译器负责将它们置零。局部变量应自己负责。如果使用了未经初始化的变量，编译器会提示。这样做的好处是：你可以摆脱因使用未初始化变量得到一个未知结果的错误。

边界检查。当数组实际上只有 *n*–1 个元素时，不可能访问到它“额外”的数组元素 *n*，这使重写未经分配的内存成为不可能。

算术运算溢出检查。C#允许在应用级或语句级检查这类操作中的溢出，使用溢出检查，当溢出发生时会出现一个异常。

A.2 基本类型

在 C#中数据类型分成两大类：一类是值类型（Value Types），一类是引用类型（Reference Types）。

A.2.1 值类型

所谓值类型就是一个包含实际数据的量，即当定义一个值类型的变量时，C#会根据它所声明的类型，以堆栈方式分配一块大小相适应的存储区域给这个变量，随后对这个变量的读或写操作就直接在这块内存区域进行。

C#中的值类型包括：简单类型、枚举类型和结构类型。

1．简单类型

简单类型是系统预置的，一共有 13 个数据类型，如表 A.1 所示。

表 A.1 C#简单类型

C#关键字	.NET CTS 类型名	说 明	范围和精度
bool	System.Boolean	逻辑值（真或假）	True, False
sbyte	System.SByte	8 位有符号整数类型	–128～127
byte	System.Byte	8 位无符号整数类型	0～255
short	System.Int16	16 位有符号整数类型	–32768～32767
ushort	System.UInt16	16 位无符号整数类型	0～65535
int	System.Int32	32 位有符号整数类型	–2147483648～2147483647

续表

C#关键字	.NET CTS 类型名	说　　明	范围和精度
uint	System.Uint32	32 位无符号整数类型	0～4294967295
long	System.Int64	64 位有符号整数类型	–9223372036854775808 ～9223372036854775807
ulong	System.UInt64	64 位无符号整数类型	0～18446744073709551615
char	System.Char	16 位字符类型	所有的 Unicode 编码字符
float	System.Single	32 位单精度浮点类型	$\pm1.5\times10^{-45}$～$\pm3.4\times10^{38}$ （大约 7 个有效十进制数）
double	System.Double	64 位双精度浮点类型	$\pm5.0\times10^{-324}$～$\pm3.4\times10^{308}$ （大约 15～16 个有效十进制数）
decimal	System.Decimal	128 位高精度十进制数类型	$\pm1.0\times10^{-28}$～$\pm7.9\times10^{28}$ （大约 28~29 个有效十进制数）

从表 A.1 可见，C#的简单数据类型可以分为整数类型（包括字符类型）、实数类型和布尔类型。

整数类型共有 9 种，它们的区别在于所占存储空间的大小、带不带符号位以及所能表示数的范围，这些是程序设计时定义数据类型的重要参数。char 类型归属于整型类别，但它与整型有所不同，不支持从其他类型到 char 类型的隐式转换。即使 sbyte、byte、ushort 这些类型的值也在 char 表示的范围之内，也不存在其隐式转换。

实数类型有 3 种，其中浮点类型 float、double 采用了 IEEE 754 格式来表示，因此浮点运算一般不产生异常。decimal 类型主要适用于财务和货币计算，它可以精确地表示十进制小数数字（如 0.001）。虽然它具有较高的精度，但取值范围较小，因此从浮点类型到 decimal 的转换可能会产生溢出异常；而从 decimal 到浮点类型的转换则可能导致精度的损失，所以浮点类型与 decimal 之间不存在隐式转换。

bool 类型表示布尔逻辑量，它与其他类型之间不存在标准转换，即不能用一个整型数表示 True 或 False，反之亦然，这点与 C/C++不同。

2．枚举类型

枚举实际上是为一组在逻辑上密不可分的整数值提供便于记忆的符号。

例如，声明一个代表颜色的枚举类型的变量：

```
enum Color {red,yellow,blue,green,black,white};
Color col;
```

枚举类型的变量在某一时刻只能取枚举中某一个元素的值。例如，col 这个表示颜色的枚举类型变量，在某一时刻只能为枚举中的一种颜色。

枚举中的每一个元素类型都是 int 型，第一个元素的值为 0，其后每一个连续的元素依次加 1 递增。也可以给元素直接赋值，如把 red 的值设为 1，其后的元素的值分别为 2、3、…。

```
enum Color {red=1,yellow,blue,green,black,white};
```

为枚举元素所赋的值的类型限于 long、int、short 和 byte 等整数类型。

3. 结构类型

结构类型是指把各种不同类型数据信息组合在一起形成的类型。结构类型的变量采用 struct 声明。例如，可以定义学生成绩表记录结构如下：

```
struct Student
{
    public string number;
    public string name;
    public int score;
}
Student stu1;
```

stu1 就是一个 Student 结构类型的变量。对结构成员的访问通过结构变量名加上访问符“.”，再跟成员名，如“stu1.name="Jacky";”。

结构类型包含的成员类型没有限制，结构类型的成员还可以是结构类型。

A.2.2 引用类型

C#中的另一大数据类型为引用类型，引用类型与值类型的区别在于：引用类型变量不直接存储所包含的值，而是实际数据的地址。C#中的引用类型有 4 种：类、数组、委托和接口。

1．类

类是面向对象编程的基本单位，其中包含数据成员、函数成员和嵌套类型的数据结构。类的数据成员有常量、字段和事件。函数成员包括方法、属性、索引指示器、运算符、构造函数和析构函数。类支持继承机制，派生类可以扩展基类的数据成员和函数成员。

2．数组

数组是同一类型数据的有序集合。有关数组的详细介绍参见 A.8 节。

3. 委托

C#的委托相当于 C/C++中的函数指针。用指针获取一个函数的入口地址，实现对函数的操作。委托与 C/C++中的函数指针不同在于，委托是面向对象的，是引用类型，因此对委托的使用要先定义、实例化，最后才调用。

定义委托使用关键字 delegate，定义一个委托：

```
delegate int SomeDelegate(int nID, string sName);
```

再实例化：

```
SomeDelegate d1 = new SomeDelegate(wr.InstanceMethod);
```

最后调用：

```
d1(5, "aaa");
```

通过委托 SomeDelegate，实现对方法 InstanceMethod 的调用，调用还必须有一个前提条件是：方法 InstanceMethod 的参数和定义 SomeDelegate 的参数一致，并且返回值为 int。方法 InstanceMethod 定义：

```
public int InstanceMethod(int nID, string sName)
```

委托的实例化中的参数既可以是非静态方法，也可以是静态方法。

4. 接口

接口中不能定义数据，只能定义方法（包括方法、属性、索引指示器等），而且只能定义方法的首部，不包括这些方法的实现。在接口中所定义的实际上是一组为所有继承者必须遵守的契约。对于使用者来说，只要知道某个类是从哪个接口继承的，就知道这个类能提供什么样的服务，以及如何调用这些提供服务的方法。

接口与接口之间可以继承，类（或结构）也可以继承接口。类与类之间只能单端继承，但类可以在继承一个基类的同时继承一个或多个接口。在继承接口的类中必须实现全部在接口中定义的方法。

接口的声明要用到 interface 关键字，声明的基本格式如下：

```
[修饰符] interface  接口名称  [：基接口表 ]
{
    // 声明接口成员
}
```

例如，

```
using System;
interface IPoint
{
      void Print(Object o);
      int X {get; set;}
      int Y { get; set;}
}
```

A.3　变量与常量

无论使用何种程序设计语言编写程序，变量和常量都是构成程序的基本元素，可以从它们的定义、命名、类型、初始化等几个方面来认识和理解变量和常量。

A.3.1　常量

1．整数常量

对于一个整数数值，默认的类型就是能保存它的最小整数类型，根据常量的值其类型可分为 int、uint、long 或 ulong。如果默认的类型不是想要的类型，可以通过在常量后面加后缀（U 或 L）来明确指定其类型。

① 在常量后面加 L 或 l（不区分大小写），表示长整型。

② 在常量后面加 U 或 u（不区分大小写），表示无符号整数。

③ 整型常量既可以采用十进制也可以采用十六进制，不加特别说明就默认为十进制。在数值前面加 0x（或 0X）则表示十六进制数，十六进制基数用 0~9、A~F（或 a~f）表示。

2．浮点常量

一般带小数点的数或用科学计数法表示的数都被认为是浮点数，它的数据类型默认为 double 类型，但也可以加后缀符表明不同的浮点格式数。

① 在数字后面加 F（f），表示是 float 类型。

② 在数字后面加 D（d），表示是 double 类型。

③ 在数字后面加 M（m），表示是 decimal 类型。

3．字符常量

字符常量简单地说就是用单引号括起的单个字符，如 'A'，它占 16 位，以无符号整型数的形式存储这个字符所对应的 Unicode 代码。这对于大多数图形字符是可行的，但对一些非图形的控制字符（如回车符）则行不通，所以字符常量的表达形式有若干种形式。

① 单引号括起的一个字符，如：'A' 。

② 十六进制的换码系列，以“\x”或“\X”开始，后面跟 4 位 16 进制数，如：'\X0041' 。

③ Unicode 码表示形式，以“\U”或“\u”开始，后面跟 4 位 16 进制数，如：'\U0041'。

④ 显式转换整数字符代码，如：(char) 65。

字符转义系列，如表 A.2 所示。

表 A.2　转义符

转 义 字 符	含　义	Unicode 码
\'	单引号	\u0027
\"	双引号	\u0022
\\	反斜线字符	\u005C
\0	空字符	\u0000
\a	警铃符	\u0007
\b	退格符	\u0008
\f	走纸换页符	\u000C
\n	换行符	\u000A
\r	回车符	\u000D
\t	水平制表符	\u0009
\v	垂直制表符	\u000B

4. 字符串常量

字符串常量是用双引号括起的零个或多个字符序列。C#支持两种形式的字符串常量，一种常规字符串，另一种逐字字符串。

（1）常规字符串

双引号括起的一串字符，可以包括转义字符。例如：

```
"Hello, world\n"
"C:\\windows\\Microsoft"          // 表示字符串 C:\windows\Microsoft
```

（2）逐字字符串

在常规的字符串前面加一个@，就形成了逐字字符串，它的意思是字符串中的每个字符

均表示本意，不使用转义字符。如果在字符串中需用到双引号，则可连写两个双引号来表示一个双引号。

例如：

```
@"C:\windows\Microsoft"          // 与 "C:\\windows\\Microsoft" 含义相同
@"He said""Hello"" to me"        // 与"He said\"Hello\" to me" 含义相同
```

5. 布尔常量

它只有两个值：true 和 false。

6. 符号常量

在声明语句中，可以声明一个标识符常量，但必须在定义标识符时就进行初始化，并且定义后就不能再改变该常量的值。

具体的格式为：

```
const　类型　标识符=初值
```

例如：

```
const　double　PI=3.14159
```

A.3.2　变量

变量是在程序的运行过程中其值可以改变的量，它是一个已命名的存储单元，通常用来记录运算中间结果或保存数据。C#中的变量必须先声明后使用。声明变量包括变量的名称、数据类型及必要时指定变量的初始值。

变量声明格式：

```
类型　标识符 [, 标识符]0+ ;
```

或

```
类型　标识符[=初值]opt [ , 标识符[=初值]opt ]0+ ;
```

标识符是变量名的符号，它的命名规则是：

```
[字母 |_|@ ]1　[字母 | 数字|_ ]0+
```

说明：

[　]$_{0+}$：表示[ ]内的内容可以出现 0 次或任意多次。

[　]$_{opt}$：表示[ ]内的内容是可选的，最多出现一次。

[　]$_{1}$：表示[ ]内的内容必须出现 1 次。

|：由竖线分隔的内容任意选择一个。

通过上述的命名规则可以看出，标识符必须以字母或下划线开头，后面可以跟字母、数字和下划线组合。例如，name、_Int、Name, x_1 等都是合法的标识符，但 C#是大小写敏感的语言，name、Name 分别代表不同的标识符，在定义和使用时要特别注意。另外变量名不能与 C# 中的关键字相同，除非标识符是以@作为前缀的。

例如：

```
int　x ;                          // 合法
float y1=0.0, y2 =1.0, y3 ;       // 合法，变量说明的同时可以设置初始数值
```

```
string  char                    // 不合法，因为 char 是关键字
string  @char                   // 合法
```

C#允许在任何模块内部声明变量，模块开始于“{”，结束于“}”。每次进入声明变量所在的模块时，则创建变量分配存储空间；离开这个模块时，则销毁这个变量，收回分配的存储空间。实际上变量只在这个模块内有效，所以称为局部变量，这个模块区域就是变量的作用域。

A.4 运算符与表达式

C#提供了大量的运算符，按需要操作数的数目来分，可以有一元运算符（如++）、二元运算符（如+）、三元运算符（如? : ）。按运算功能来分，基本的运算符可以分为：算术运算符、关系运算符、逻辑运算符、位运算符、赋值运算符、条件运算符、其他（分量运算符、下标运算符等）。

本节主要介绍前 6 种运算符，以及这些运算符的优先级、结合性等。

A.4.1 算术运算符

算术运算符作用的操作数类型可以是整型，也可以是浮点型，运算符如表 A.3 所示。

表 A.3 算术运算符

运 算 符	含 义	示例（假设 x、y 是某一数值类型的变量）
+	加	x + y； x+3；
–	减	x – y； y–1；
*	乘	x * y； 3*4；
/	除	x / y； 5/2; 5.0/2.0；
%	取模	x % y； 11%3； 11.0 % 3;
++	递增	++x； x++；
––	递减	––x； x––；

其中，“+–*/ ”运算与一般代数意义及其他语言相同，但需要注意的是：当“/”作用到的两个操作数都是整型数据类型时，其计算结果也是整型。

“%”取模，即获得整数除法运算的余数，所以也称取余。

“++”递增和“– –”递减运算符是一元运算符，它作用的操作数必须是变量，不能是常量或表达式。它既可出现在操作数之前（前缀运算），也可出现在操作数之后（后缀运算），前缀和后缀有共同之处，也有很大区别。

A.4.2 关系运算符

关系运算符用来比较两个操作数的值，运算结果为 bool 类型的值（true 或 false）。

C# 中，简单类型和引用类型都可以通过= =或!=来比较它们的数据内容是否相等。对简单类型，比较的是它们数据值。而对引用类型来说，由于它的内容是对象实例的引用，所以

若相等，则说明这两个引用指向同一个对象实例。如果要测试两个引用对象所代表的内容是否相等，通常会使用对象本身所提供的方法，如 Equals( )。

如果操作数是 string 类型，则在下列两种情况下视为两个 string 值相等。

① 两个值均为 null。

② 两个值都是对字符串实例的非空引用，这两个字符串不仅长度相同，并且每个对应的字符位置上的字符也相同。

关系比较运算>、>=、<、<=以顺序作为比较的标准，所以它要求操作数的数据类型只能是数值类型，即整型数、浮点数、字符及枚举等类型。

bool 类型的值只能比较是否相等，不能比较大小。因为 true 和 false 值没有大小之分，如表达式 true > flase 等在 C# 中是没有意义的。

A.4.3　逻辑运算符

逻辑运算符是用来对两个 bool 类型的操作数进行逻辑运算的，运算的结果也是 bool 类型，如表 A.4 所示。

表 A.4　逻辑运算符

运　算　符	含　　义
&	逻辑与
|	逻辑或
^	逻辑异或
&&	短路与
||	短路或
!	逻辑非

运算符&&和||的操作结果与&和|一样，但它们的短路特征，使代码的效率更高。所谓短路就是在逻辑运算的过程中，如果计算第一个操作数时，就能得知运算结果，就不会再计算第二个操作数。

例如：

```
int  x，y；
bool  z；
x = 1；  y = 0；
z = ( x >1) & (++ y >0 )          // z 的值为 false，y 的值为 1
z = ( x >1) && (++ y >0 )         // z 的值为 false，y 的值为 0
```

逻辑非运算符!是一元运算符，它对操作数进行非运算，即真/假值互为非（反）。

A.4.4　位运算符

位运算符主要分为逻辑运算和移位运算，它的运算操作直接作用于操作数的每一位，所以操作数的类型必须是整数类型，不能是 bool、float 或 double 等类型。

位运算符如表 A.5 所示。借助这些位运算符可以完成对整型数的某一位测试、设置，以及对一个数的位移动等操作，这对许多系统级程序设计是非常重要。

表 A.5　位运算符

运　算　符	含　　义
&	按位与
|	按位或
^	按位异或
~	按位取反
>>	右移
<<	左移

A.4.5　赋值运算符

赋值运算符的一般格式为：

```
Var = Expression
```

赋值运算符左边的称为左值，右边的称为右值。右值是一个与左值类型兼容的表达式，它可以是常量、变量或表达式。左值必须是一个已定义的变量或对象，因为赋值运算的操作就是将右值复制到左值，因此左值必须是内存中已分配的实际物理空间。

赋值运算的结果是：其值是右边表达式的值，类型是左值类型。如果左值和右值的类型不一致，在兼容的情况下，则需要进行自动转换（隐式转换）或强制类型转换（显式类型转换）。一般的原则是，从占用内存较少的短数据类型向占用内存较多的长数据类型赋值时，可以不做显式的类型转换，C#会进行自动类型转换；反之当从较长的数据类型向占用较少内存的短数据类型赋值时，则必须做强制类型转换。

A.4.6　条件运算符

条件运算符 ?: 是 C# 中唯一的三元运算符，其形式为：

```
Exp1  ? Exp2 : Exp3
```

其中表达式 Exp1 的运算结果必须是一个 bool 类型值，表达式 Exp2 和 Exp3 可以是任意数据类型，但它们返回的数据类型必须一致。

条件运算符的运算过程是：首先计算 Exp1 的值，如果其值为 true，则计算 Exp2 值，这个值就是整个表达式的结果；否则，取 Exp3 的值作为整个表达式的结果。

例如：

```
z = x > y ? x : y ;          // z 的值就是 x、y 中较大的一个值
z = x >=0 ? x : -x ;         // z 的值就是 x 的绝对值
```

A.4.7　运算符的优先级与结合性

当一个表达式含有多个运算符时，C#编译器需要知道先做哪个运算，这就是所谓的运算符优先级，它控制各运算符的运算顺序。

表 A.6 列出了 C#运算符的优先级与结合性，其中顶部的优先级较高。

表 A.6　运算符的优先级与结合性

类　别	运　算　符	结　合　性
初等项	. ( ) [ ] new typeof checked unchecked	从左到右
一元后缀	++ ––	从右到左
一元前缀	++ –– + - ! ~ (T) (表达式)	从右到左
乘法	* / %	从左到右
加法	+ -	从左到右
移位	<< >>	从左到右
关系和类型检测	< > <= >= is as	从左到右
相等	== !=	从左到右
逻辑与	&	从左到右
逻辑异或	^	从左到右
逻辑或	|	从左到右
条件与	&&	从左到右
条件或	||	从左到右
条件	? :	从右到左
赋值	= *= /= %= += -= <<= >>= &= ^= |=	从右到左

A.5　分支语句

分支语句就是条件判断语句，它能让程序在执行时根据特定条件是否成立而选择执行不同的语句块。C#提供两种分支语句结构：if 语句和 switch 语句。

A.5.1　if语句

if 语句是最常用的选择语句，用来判断是否满足给定的条件，根据判定的结果决定执行给出的两种操作之一。

if 语句有两种基本形式。

```
if  (布尔表达式) 语句;
```

当布尔表达式的值为真，则执行 if 后面的内嵌语句，为假则继续执行 if 语句的后继语句。

```
if (布尔表达式) 语句 1 else 语句 2;
```

当布尔表达式的值为真，则执行内嵌语句 1，否则执行内嵌语句 2。

如果程序的逻辑判断关系比较复杂，通常还可以采用条件判断嵌套语句。具体形式如下：

```
if  (布尔表达式)     {     if (布尔表达式)  语句 1   else  语句 2; }
else { if (布尔表达式)  语句 3   else  语句 4;}
```

此时每一条 else 与离它最近且没有其他 else 与之对应的 if 相搭配。

A.5.2 switch语句

switch 语句是一个多分支结构的语句，它所实现功能与 if else 结构很相似，但在大多数情况下，switch 语句表达方式更直观、简单、有效。

形式：

```
switch  (表达式)
{
    case  常量 1：
        语句序列 1；                    // 由零个或多个语句组成
        break ；
    case  常量 2：
        语句序列 2；
        break ；
    ......
    [ default：                         // default 是任选项，可以不出现
        语句序列 n；
        break ； ]
}
```

switch 语句的执行流程是，首先计算 switch 后的表达式，然后将结果值一一与 case 后的常量值比较，如果找到相匹配的 case，程序就执行相应的语句序列，直到遇到跳转语句（break），switch 语句执行结束；如果找不到匹配的 case，就归结到 default 处，执行它的语句序列，直到遇到 break 语句为止；如果没有 default，则不执行任何操作。

C#的 switch 语句需要注意以下几点：

① switch 语句的表达式必须是整数类型，如 char、sbyte、byte、ushort、short、uint、int、ulong、long、string、枚举类型，case 常量必须与表达式类型相兼容，case 常量的值必须互异，不能有重复。

② 将与某个 case 相关联的语句序列接在另一个 case 语句序列之后，是错误的，这称为“不穿透”规则，所以需要跳转语句结束这个语句序列，通常选用 break 语句作为跳转，也可以用 goto 转向语句。“不穿透”规则是 C# 对 C、C++、Jave 这类语言中的 switch 语句的一个修正，这样做的好处是：一是允许编译器对 switch 语句做优化处理时可自由地调整 case 的顺序；二是防止程序员不经意地漏掉 break 语句而引起错误。

③ 虽然不能让一个 case 的语句序列穿透到另一个 case 语句序列，但是可以有两个或多个 case 前缀指向相同的语句序列。

A.6 循环语句

循环语句是指在一定条件下，重复执行一组语句，它是程序设计中的一个非常重要、基本的方法。C#提供了 4 种循环语句：while、do_while、for 和 foreach。

A.6.1　while 语句

语法形式：

```
while (条件表达式)
    循环体语句;
```

如果条件表达式为真（true），则执行循环体语句。while 语句执行流程如图 A.1 所示。

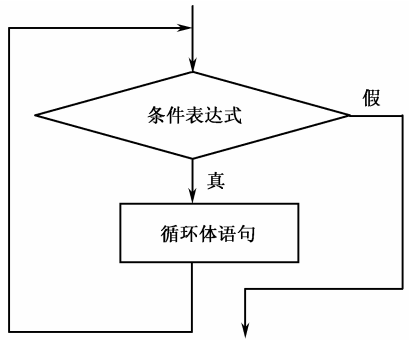


图 A.1　while 语句执行流程图

A.6.2　do_while 语句

语法形式：

```
do
{
    循环体语句;
}while (条件表达式)
```

该循环首先执行循环体语句，再判断条件表达式。如果条件表达式为真（true），则继续执行循环体语句。do_while 循环语句执行流程如图 A.2 所示。

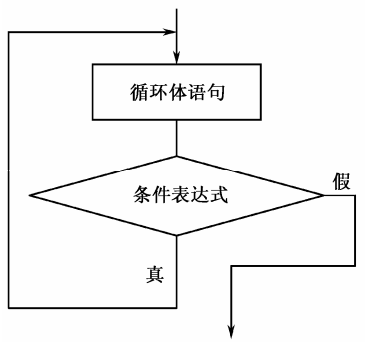


图 A.2　do_while 语句执行流程图

while 语句与 do_while 语句很相似，它们的区别在于 while 语句的循环体有可能一次也不执行，而 do_while 语句的循环体至少执行一次。

A.6.3 for 语句

在事先知道循环次数的情况下，使用 for 语句比较方便。for 语句的格式为：

```
for(表达式 1;表达式 2;表达式 3)  语句
```

其中表达式 1 为循环控制变量初始化，循环控制变量可以有一个或多个，若有多个则用逗号隔开；表达式 2 为循环控制条件；表达式 3 按规律改变循环控制变量的值。三个表达式都是可选的，省略某个表达式时，其后的分号“;”不能省，三个表达式都省略的情况如：for(;;){语句}

for 语句的执行顺序如下：

① 按顺序将表达式 1 执行一遍，为循环控制变量赋初值；

② 测试表达式 2 是否为真；

③ 若没有表达式 2 或表达式 2 为真，则执行内嵌语句一遍，按表达式 3 的规律改变循环控制变量的值，回到第②步执行；

④ 若表达式 2 不满足，则 for 循环终止。

图 A.3 说明了 for 语句的执行流程。

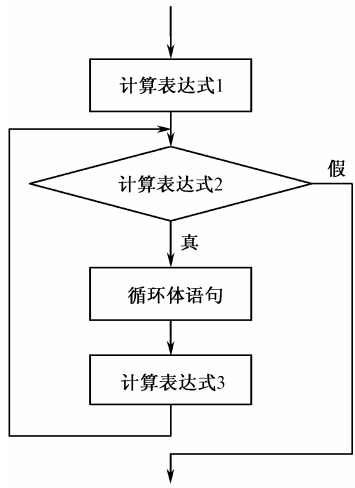


图 A.3 for 语句执行流程图

A.6.4 foreach 语句

foreach 语句是 C# 中新引入的，它表示收集集合中的各元素，并针对各元素执行内嵌语句。foreach 语句的一般形式为：

```
foreach (类型  标识符  in  集合表达式 ) 语句;
```

其中，标识符是 foreach 循环的迭代变量，它只在 foreach 语句中有效，并且是一个只读局部变量，也就是说在 foreach 语句中不能改写这个迭代变量。它的类型应与集合的基本类型一致。集合表达式是被遍历的集合，如数组。

在 foreach 语句执行期间，迭代变量按集合元素的顺序依次将其内容读入。

A.7　跳转语句

跳转语句用于改变程序的执行流程，转移到指定处。C#中有 4 种跳转语句：continue 语句、break 语句、return 语句、goto 语句。它们具有不同的含义，用于特定的上下文环境中。

A.7.1　continue语句

语法形式：

```
continue ;
```

continue 语句只能用于循环语句中，作用是结束本轮循环，不再执行余下的循环体语句。对 while 和 do_while 结构的循环，在 continue 执行之后就立刻测试循环条件，以决定循环是否继续下去；对 for 结构循环，在 continue 执行之后，先求表达式 3（即循环增量部分），然后再测试循环条件。通常它会和一个条件语句结合起来用，不会是独立的一条语句，也不会是循环体的最后一条语句，否则没有任何意义。

如果 continue 语句陷于多重循环结构之中，它只对包含它最内层的循环有效。

A.7.2　break语句

语法形式：

```
break;
```

break 语句只能用于循环语句或 switch 语句中，如果在 switch 语句中执行到 break 语句，则立刻从 switch 语句中跳出，转到 switch 语句的下一条语句；如果在循环语句执行到 break 语句，则会导致循环立刻结束，跳转到循环语句的下一条语句。不管循环有多少层，break 语句只能从包含它的最内层循环跳出一层。

A.7.3　return语句

语法形式：

```
return;
```

或

```
return　表达式;
```

return 语句出现在一个方法内，在方法中执行到 return 语句时，程序流程转到调用这个方法处。如果方法没有返回值（返回类型修饰为 void），则使用 return 返回；如果方法有返回值，那么使用 return 表达式格式，其后面跟的表达式就是方法的返回值。

A.7.4　goto语句

goto 语句可以将程序的执行流程从一个地方转移到另一个地方，非常灵活，但正因为它太灵活，所以容易造成程序结构混乱的局面，应该有节制、合理地使用 goto 语句。

语法形式：

```
goto　标号;
```

其中标号就是定位在某一语句之前的一个标识符，称为标号语句，格式如下：

```
标号：语句;
```

它给出 goto 语句转向的目标。值得注意的是，goto 语句不能使控制转移到另一个语句块内部，更不能转到另一个函数内部。

另外 goto 语句如果用在 switch 语句中，它的格式是：

```
goto  case 常量;
goto  default ;
```

它只能在本 switch 语句中从一种情况转向另一种情况。

A.8 数组

数组是一种包含若干变量的数据结构，这些变量都具有相同的数据类型，并且排列有序，因此可以用一个统一的数组名和下标唯一地确定数组中的元素。C# 中的数组主要有三种形式：一维数组、多维数组和不规则数组。

A.8.1 数组的定义

一般而言，数组都必须先声明后使用。在 C/C++ 这类语言中，数组在声明时，就要明确数组的元素个数，由编译器分配存储空间。但在 C# 中，数组是一个引用型类型，声明数组时，只是预留一个存储位置以引用将来的数组实例，实际的数组对象是通过 new 运算符在运行时动态产生的。因此在数组声明时，不需要给出数组的元素个数。

1．一维数组

（1）一维数组声明语法形式

```
type  [ ]  arrayName ;
```

其中，type 可以是 C#中任意的数据类型；[ ]表明后面的变量是一个数组类型，必须放在数组名之前；arrayName 是数组名，遵循标识符的命名规则。

例如：

```
int [ ] a1;              // a1 是一个含有 int 类型数据的数组
double [ ] f1;           // f1 是一个含有 double 类型数据的数组
string [ ] s1;           // s1 是一个含有 string 类型数据的数组
```

（2）创建数组对象

用 new 运算符创建数组实例，有 2 种基本形式。

声明数组和创建数组分别进行：

```
type [ ] arrayName ;                  // 数组声明
arrayName = new type [size];          // 创建数组实例
```

其中，size 表明数组元素的个数。

声明数组和创建数组实例也可以合在一起：

```
type [ ] arrayName = new type [size] ;
```

例如：

```
int [ ] a1;
    a1 = new int [10];                    // a1 是一个有 10 个 int 类型元素的数组
    string [ ] s1 = new string [5];       // s1 是一个有 5 个 string 类型元素的数组
```

2．多维数组

（1）多维数组声明语法形式

```
type  [ ,  ,  , ] arrayName ;
```

多维数组是指能用多个下标访问的数组。在声明时方括号内加逗号，就表明是多维数组，有 *n* 个逗号，就是 *n*+1 维数组。

例如：

```
int [ , ] score;              // score 是一个 int 类型的二维数组
float [ , , ] table;          // table 是一个 float 类型的三维数组
```

（2）创建数组对象

声明数组和创建数组分别进行：

```
type [ ,  ,  , ] arrayName ;                          //数组声明
arrayName = new type [size1, size2, size3];
//创建数组实例，size1、size2、size3 分别表明多维数组每一维的元素个数
```

声明数组和创建数组实例也可以合在一起：

```
type [,  ,  , ] arrayName = new type [size1, size2, size3] ;
```

例如：

```
int [, ] score ;
score = new int [3, 4] ;                  //score 是一个 3 行 4 列的二维数组
float [ , , ] table=new float [2, 3, 4]   //table 是一个三维数组，每维分别是 2、3、4
```

（3）不规则数组

一维数组和多维数组都属于矩形数组，而 C#所特有的不规则数组是数组的数组，它的内部每个数组的长度可以不同，就像一个锯齿形状。

① 不规则数组声明。

```
type [ ] [ ] [ ] arrayName ;
```

方括号[ ]的个数与数组的维数相关。

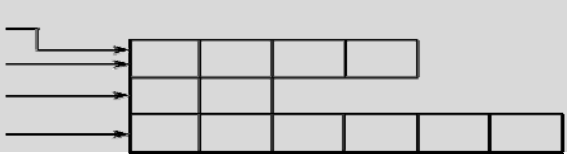
例如：

```
int  [ ] [ ] jagged ;                     // jagged 是一个 int 类型的二维不规则数组
```

② 创建数组对象。

以二维不规则数组为例：

```
int [ ] [ ] jagged;
jagged = new int [3][ ];
jagged[0] = new int [4];
jagged[1] = new int [2];
jagged[2] = new int [6];
```



A.8.2 数组的初始化

在用 new 运算符生成数组实例时，若没有对数组元素初始化，则取它们的默认值，数值型变量默认值为 0，引用型变量默认值为 null。当然数组也可以在创建时按照自己的需要进行初始化，需要注意的是，初始化时不论数组的维数是多少，都必须显示地初始化所有数组元素，不能进行部分初始化。

1．一维数组初始化

语法形式 1：

```
type [ ] arrayName = new type [size] { val1, val2, …,valn };
```

数组声明与初始化同时进行时，size 也就是数组元素的个数必须是常量，而且应该与大括号内的数据个数一致。

语法形式 2：

```
type [ ] arrayName = new type [ ] { val1, val2, …,valn };
```

省略 size，由编译系统根据初始化表中的数据个数，自动计算数组的大小。

语法形式 3：

```
type [ ] arrayName = { val1, val2, …,valn };
```

数组声明与初始化同时进行，还可以省略 new 运算符。

语法形式 4：

```
type [ ] arrayName ;
arrayName = new type [size] { val1, val2, …,valn };
```

把声明与初始化分开在不同的语句中进行时，size 同样可以省略，也可以是一个变量。例如，以下数组初始化实例都是等同的。

```
int [ ] nums = new int [10] {0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int [ ] nums = new int [ ] {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
int [ ] nums = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int [ ] nums ; nums = new int [10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

2．多维数组初始化

多维数组初始化是通过将对每维数组元素设置的初始值放在各自的一个大花括号内完成，以最常用的二维数组为例来讨论。

语法形式 1：

```
type [ , ] arrayName = new type [size1, size2 ] {{ val11, val12, …,val1n },
                                                 { val21, val22, …,val2n }, …, { valm1, valm2, …,valmn };
```

数组声明与初始化同时进行，数组元素的个数是 size1×size2，数组的每一行分别用一个花括号括起来，每个花括号内的数据就是这行的每一列元素的值，初始化时的赋值顺序按矩阵的“行”存储原则。

语法形式 2：

```
type [ ] arrayName = new type [ , ] {{ val11, val12, …,val1n },
                                    { val21, val22, …,val2n }, … { valm1, valm2, …,valmn };
```

省略 size，由编译系统根据初始化表中花括号{}的个数确定行数，再根据{}内的数据确定列数，从而得出数组的大小。

语法形式 3：

```
type [ , ] arrayName ={{ val11, val12, …,val1n },
                       { val21, val22, …,val2n }, …, { valm1, valm2, …,valmn };
```

数组声明与初始化同时进行，还可以省略 new 运算符。

语法形式 4：

```
type [ , ] arrayName ;
arrayName = new type [size1, size2] {{ val11, val12, …,val1n },
                                    { val21, val22, …,val2n }, …, { valm1, valm2, …,valmn };
```

把声明与初始化分开在不同的语句中进行时，size1、size2 同样可以省略，但也可以是变量。

例如，以下数组初始化实例都是等同的。

```
int [, ] a = new int [3,4] {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} };
int [, ] a = new int [, ] {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} };
int [, ] a = {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} };
int [ ] a ; a = new int [3, 4] {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} };
```

3．不规则数组初始化

下面以二维不规则数组为例来讨论。

不规则数组是一个数组的数组，所以它的初始化通常是分步骤进行的。

```
type [ ] [ ] arrayName = new type [ size] [ ];
```

size 可以是常量或变量，后面一个中括号[ ]内是空着的，表示数组的元素还是数组，且每个数组的长度是不一样，需要单独再用 new 运算符生成。

```
arrayName[0] = new type [size0] { val1, val2, …, valn1};
arrayName[1] = new type [size1] { val1, val2, …, valn2};
  ……
```

例如：

```
char [ ] [ ] st1 = new char [3][ ];                    // st1 是由 3 个数组组成的数
st1[0] = new char [ ] {'S', 'e', 'p', 't', 'e', 'm', 'b', 'e', 'r' }
st1[1] = new char [ ] {'O', 'c', 't', 'o', 'b', 'e', 'r'}
st1[2] = new char [ ] {'N', 'o', 'v', 'e', 'm', 'b', 'e', 'r' };
```

A.8.3　数组元素的访问

一个数组具有初值时，就可以像其他变量一样被访问，既可以取数组元素的值，也可

以修改数组元素的值。在 C#中是通过数组名和数组元素的下标来引用数组元素。

1．一维数组的引用

一维数组的引用语法形式：

```
数组名[下标]
```

下标是数组元素的索引值，实际上就是要访问的那个数组元素在内存中的相对位移。相对位移是从 0 开始的，所以下标的值从 0 到数组元素的个数–1 为止。

2．多维数组的引用

多维数组的引用语法形式：

```
数组名[下标 1, 下标 2, …,下标 n]
```

3．不规则数组的引用

语法形式：

```
数组名[下标 1][下标 2]…[下标 n]
```

A.9 综合应用实例

【例 A-1】 扑克牌游戏。用计算机模拟洗牌，分发给 4 个玩家，并将 4 个玩家的牌显示出来。

【基本思路】

一维数组 Card 存放 52 张牌（不考虑大、小王），二维数组 Player 存放 4 个玩家的牌。

用三位整数表示一张扑克牌，最高位表示牌的种类，后两位表示牌号。例如：

101，102，…，113 分别表示红桃 A，红桃 2，…，红桃 K；

201，202，…，213 分别表示方块 A，方块 2，…，方块 K；

301，302，…，313 分别表示梅花 A，梅花 2，…，梅花 K；

401，402，…，413 分别表示黑桃 A，黑桃 2，…，黑桃 K；

【创建控制台应用程序】

新建控制台应用程序方法如下。

选择“文件”→“新建”→“项目”菜单项，弹出的“新建项目”对话框，在“项目类型”中选择“Visual C#”下的“Windows”选项，在“模板”中选择“控制台应用程序”模板。名称设置为应用程序的名称，这里设置为“TestCard”，单击“确定”按钮完成控制台应用程序的创建。

【程序清单】

```
using System;
class TestCard
{
    static void Main(string[] args)
    {
        int i, j, temp;
```

```
Random Rnd = new Random();
int k;
int [] Card = new int [52];
int [,] Player=new int [4,13];
for (i=0; i<4; i++)                          //52 张牌初始化
    for (j=0; j<13; j++)
        Card[i*13+j]=(i+1)*100+j+1;
Console.Write ("How many times for card: ");
string s=Console.ReadLine ();
int times=Convert.ToInt32 (s);
for (j=1;j<=times; j++)
    for (i=0;i<52;i++)
    {
        k=Rnd.Next (51-i+1)+i;               //产生 i 到 52 之间的随机数
        temp=Card[i];
        Card[i]=Card[k];
        Card[k]=temp;
    }
k=0;
for (j=0;j<13;j++)                           //52 张牌分发给 4 个玩家
    for (i=0;i<4;i++)
        Player[i,j]=Card[k++];
for (i=0;i<4;i++)                            //显示 4 个玩家的牌
{
    Console.WriteLine ("玩家{0}的牌：",i+1);
    for (j=0;j<13;j++)
    {
        k=(int)Player[i,j]/100;              //分离出牌的种类
        switch (k)
        {
            case 1:                          //红桃
                s=Convert.ToString ('\x0003');
                break;
            case 2:                          //方块
                s=Convert.ToString ('\x0004');
                break;
            case 3:                          //梅花
                s=Convert.ToString ('\x0005');
                break;
            case 4:                          //黑桃
                s=Convert.ToString ('\x0006');
                break;
```

```
                    }
                    k=Player[i,j]%100;                    //分离出牌号
                    switch (k)
                    {
                        case 1:
                            s=s+"A";
                            break;
                        case 11:
                            s=s+"J";
                            break;
                        case 12:
                            s=s+"Q";
                            break;
                        case 13:
                            s=s+"K";
                            break;
                        default:
                            s=s+Convert.ToString (k);
                            break;
                    }
                    Console.Write (s);
                    if (j<12)
                        Console.Write (", ");
                    else
                        Console.WriteLine (" ");
                }
            }
            Console.Read ();
        }
}
```

运行结果如图 A.4 所示。

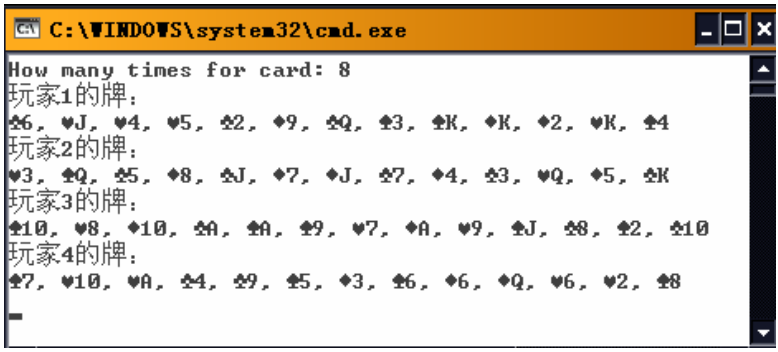
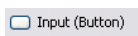
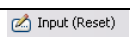
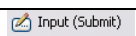
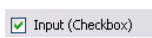
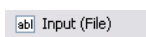
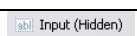
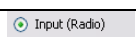
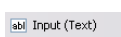
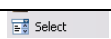
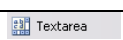


图 A.4　扑克牌游戏程序的运行结果

附录B　ASP.NET常用控件列表

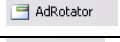
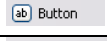
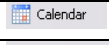
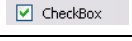
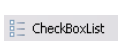
B.1　HTML服务器控件（见表B.1）

表 B.1

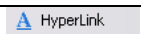
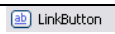
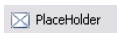
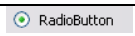
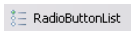
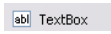
控　件　名	控 件 图 标	对应的 HTML 标记	用　途
Input(Button)		<input type="button">	可以以编程方式访问服务器上的 HTML <button> 标记
Input(Reset)		<input type="Reset">	用户可以利用该控件创建重置按钮
Input(Submit)		<input type="Submit">	用户可以利用该控件创建提交按钮
Image		<img> </img>	用于在窗体页上显示图像
Input(CheckBox)		<input type="checkbox">	创建使用户可以选择 true 或 false 状态的复选框控件
Input(File)		< input type="file">	可使用该控件方便地设计页，该页允许用户将二进制文件或文本文件从浏览器上载到在 Web 服务器上指定的文件夹中
Input(Hidden)		<input type="Hidden">	将信息存储在窗体上不可查看的控件中
Input(Radio)		<input type="radio">	创建单选按钮
Input(Text)		<input type="text"> 和 <input type="password">	创建单行文本框以接收用户输入
Select		<select> </select>	用来创建列表控件和下拉列表
Table		<table> </table>	允许以编程方式访问 HTML 的表元素
TextArea		<textarea> </textarea>	创建多行文本框

B.2　Web服务器控件（见表B.2）

表 B.2

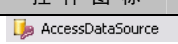
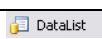
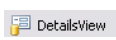
控　件　名	控 件 图 标	语 法 标 记	用　途
AdRotator		<asp:AdRotator/>	提供在移动页上显示随机选择的广告的功能
Button		<asp: Button/>	在网页上显示下压按钮控件
BulletedList		<asp:BulletedList/>	该控件生成一个采用项目符号格式的项列表
Calendar		<asp:Calendar/>	显示日历以允许用户选择日期
CheckBox		< asp:CheckBox/>	显示允许用户选择 true 或 false 条件的复选框
CheckBoxList		< asp:CheckBoxList/>	创建一组复选框。该列表控件使得可以使用数据绑定创建复选框

续表

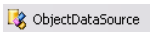
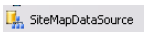
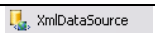
控 件 名	控 件 图 标	语 法 标 记	用 途
DropDownList		< asp: DropDownList/>	允许用户或者从列表中选择或者输入文本
FileUpload		<asp:FileUpload />	显示一个文本框控件和一个浏览按钮，使用户可以选择要上载到服务器的文件
Hyperlink		< asp: hyperlink/>	创建 Web 导航链接
Image		< asp: Image/>	显示图像
ImageButton		< asp: ImageButton/>	与 Button 控件相同，但包含图像而不是文本
Label		< asp: Label/>	显示用户无法直接编辑的文本
LinkButton		< asp: LinkButton />	与 Button 控件相同，但具有超链接的外观
ListBox		< asp: ListBox />	显示选择列表，列表可以允许多重选择（可选）
Literal		<asp:Literal/>	呈现为文本（无 HTML 标记），从而提供一种将文本放入服务器代码所产生的页面中的简便方法
MultiView		<asp:MultiView/>	用做一组 View 控件的容器控件
Panel		<asp:Panel/>	在窗体上创建无边框间隔区域，用做其他控件的容器
PlaceHolder		<asp:PlaceHolder/>	提供一个容器，以存储动态添加到 Web 页中的服务器控件。它不生成任何可见输出，只用做 Web 页上其他控件的容器
RadioButton		< asp: RadioButton />	显示可以打开或关闭的单个按钮
RadioButtonList		<asp:RadioButtonList/>	创建一组单选按钮。在该组中，只能选择一个按钮
Table		< asp: Table />	创建表。可以在服务器代码中添加行和列
TextBox		< asp: TextBox />	显示设计时输入的文本，用户可以在运行时编辑此文本，或者编程更改此文本
View		<asp:View/>	该控件作为 MultiView 控件中的一组控件的容器
Wizard		<asp:Wizard /><wizardsteps> <asp:WizardStep/></wizardsteps>	提供导航和用户界面（UI），以跨多个步骤收集相关数据
XML		<asp:Xml/>	显示来自 XML 文件或流的信息，还可以用于应用 XSLT 转换

B.3 数据控件（见表B.3）

表 B.3

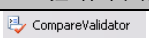
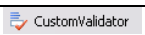
控 件 名	控 件 图 标	用 途
AccessDataSource		表示一个使用 Microsoft Access 数据库的数据源控件
DataList		显示使用模板的项的数据绑定列表控件
DetailsView		将数据源中单条记录的值显示在表中，该表的每一数据行表示该记录的一个字段。使用该控件可以编辑、删除和插入记录
FormView		使用用户定义的模板显示数据源中单个记录的值。使用该控件可以编辑、删除和插入记录

续表

控 件 名	控 件 图 标	用　途
GridView		在表中显示数据源的值，其中每列表示一个字段，每行表示一条记录。该控件允许选择和编辑这些项，以及对它们进行排序
ObjectDataSource		表示合并参数后解析方法多层 Web 应用程序结构中的数据绑定控件提供数据的业务对象
Repeater		一个数据绑定列表控件，允许通过为列表中显示的每一项重复指定的模板来自定义布局
SiteMapDataSource		提供了一个数据源控件，Web 服务器控件及其他控件可使用该控件绑定到分层的站点地图数据
SqlDataSource		表示数据绑定控件的 SQL 数据库
XmlDataSource		表示数据绑定控件的 XML 数据源

B.4　验证控件（见表B.4）

表 B.4

控 件 名	控 件 图 标	用　途
CompareValidator		将关联的输入控件的值与另一个输入控件或常量值进行比较
CustomValidator		允许自定义代码在客户端或服务器上执行验证
RangeValidator		检查输入控件的值是否在指定的值范围内
RegularExpressionValidator		验证关联的输入控件的值是否与正则表达式所指定的模式匹配
RequiredFieldValidator		使关联的输入控件成为必选字段
ValidationSummary		显示在 Web 页、消息框或这两者内部列出的所有验证错误的摘要

附录C　安装配置ASP.NET开发和运行环境

C.1　配置运行环境IIS

ASP.NET 正式版对操作系统要求：Windows 2000 以上版本，IIS 5.0 以上版本，浏览器 IE 5.5 以上版本。

C.1.1　安装IIS

如果操作系统是 Windows 2000/2003 Server，则 IIS 5.0/IIS 6.0 已经是默认安装上的；如果操作系统是 Windows 2000 professional，则需要安装 IIS 5.0；如果操作系统是 Windows XP，则需要安装 IIS 5.1。

（1）从“控制面板”的“添加或删除程序”选项中选择“添加/删除 Windows 组件”，选择“Internet 信息服务（IIS）”，然后单击“详细信息”按钮，把所有子组件全选中，连续单击“确定”、“下一步”按钮。根据要求插入系统安装盘，安装完毕后，可以在浏览器中输入 http://localhost 进行测试。如果安装成功的话，将会出现环境界面，IIS 运行环境就建立好了。

（2）运行“控制面板”的“管理工具”中的“Internet 信息服务”，即可打开 IIS 服务管理界面，在界面中可以设置默认站点的位置，在默认站点内建立虚拟目录，设置站点首页等。详细参考 IIS 管理手册。

C.1.2　安装.NET FrameWork

安装完 IIS 以后，已经能够执行 ASP 脚本了。为了支持 ASP.NET 3.5 脚本，还必须安装对应的.NET FrameWork 3.5 版本。本版本可在微软网站上下载，下载后，运行安装即可。安装好以后，ASP.NET 的运行环境就建立好了。

C.1.3　测试ASP.NET的运行环境

ASP.NET 文件的扩展名为 aspx，和其他脚本语言一样也是纯文本文件。新建命名为“default.aspx”文件，内容如下：

```
<%@ Page Language="C#" %>
<% Response.Write("Hello ASP.NET 3.5"); %>
```

把其存放到 C:\Inetpub\wwwroot 目录下，然后在浏览器地址栏中输入 http://localhost/default.aspx。如果运行后网页显示“Hello ASP.NET 3.5”，则表示 ASP.NET 3.5 运行环境安装成功。

C.2　安装运行环境Visual Studio 2008

虽然用户可以从微软网站上得到免费的 C#编译器，.NET 框架的运行时也可以免费获得，但是为了有效地进行 Windows Form、ADO.NET 和 ASP.NET 的应用开发，还是应该安装和配置 Visual Studio 集成开发环境。本书的很多范例也在 Visual Studio 2008 集成开发环境下运行。

为了更好地使用本书内容，操作系统可以选择 Windows 2000 Professional、Windows 2000 /XP/2003。为了更好地进行本书的学习，我们选择 Visual Studio 2008 Professional 版本，读者也可以根据自己的需求选择其他版本安装。

C.2.1　安装Visual Studio 2008 及产品文档

要顺利地安装与运行 Visual Studio 2008 Professional，系统配置要求如下。

① 600 MHz Pentium 处理器，推荐采用 1 GHz Pentium 处理器。

② RAM 最低要求 192 MB，推荐 256 MB。

③ 硬盘：

不安装 MSDN：安装驱动器上要有 2 GB 可用空间，系统驱动器上要有 1GB 可用空间。

安装 MSDN：在完全安装 MSDN 的安装驱动器上要有 3.8 GB 的可用空间，在进行默认 MSDN 安装的安装驱动器上要有 2.8 GB 的可用空间。系统驱动器上要有 1 GB 可用空间。

④ 显示器最低要求 800×600，256 色；推荐 1024×768 真彩色，32 位。

具体安装过程如下：

（1）关闭所有打开的应用程序，插入 Visual Studio 2008 安装光盘，或者从虚拟光驱载入安装镜像文件，光盘自动运行，弹出安装对话框，如图 C.1 所示。

图 C.1　安装对话框

（2）选择安装 Visual Studio 2008，进入安装向导，此时安装向导正在加载安装组件，如图 C.2 所示。

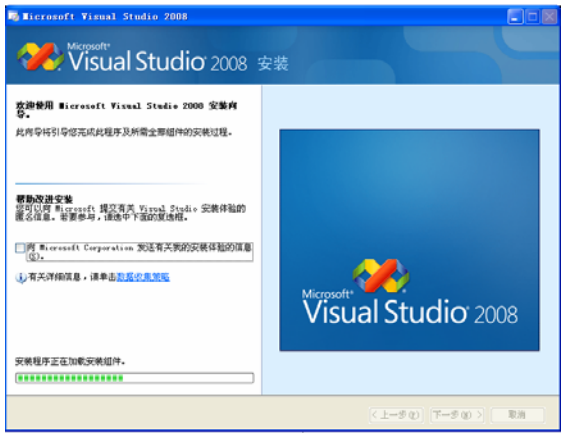


图 C.2　安装向导

（3）安装程序对已安装的组件进行扫描。在进行下一步操作前，退出所有应用程序。接受安装协议，并输入产品密钥和用户名称，然后单击“下一步”按钮，如图 C.3 所示。

图 C.3　接受许可协议

（4）进入安装功能向导，在这里可以选择安装类型，如果不希望按默认值安装，可以选择自定义，这个选项可以自由定制安装位置和选择安装组件，如图 C.4 所示。

（5）选择需要安装的组件，并自定义安装路径，单击“安装”按钮开始安装，如图 C.5 所示。

（6）安装向导将根据先前的定制要求开始安装，整个过程大约需要 20 分钟左右的时间，如图 C.6 所示。

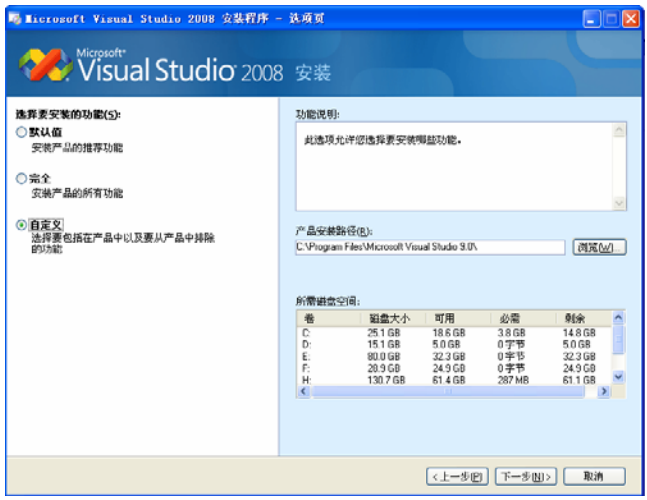


图 C.4　选择安装功能

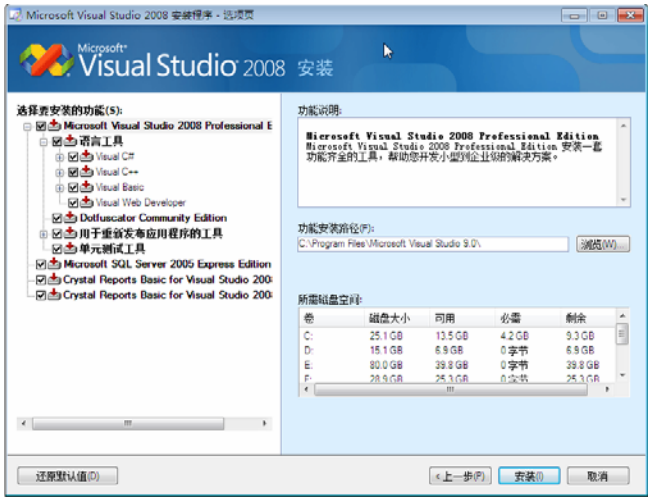


图 C.5　自定义安装

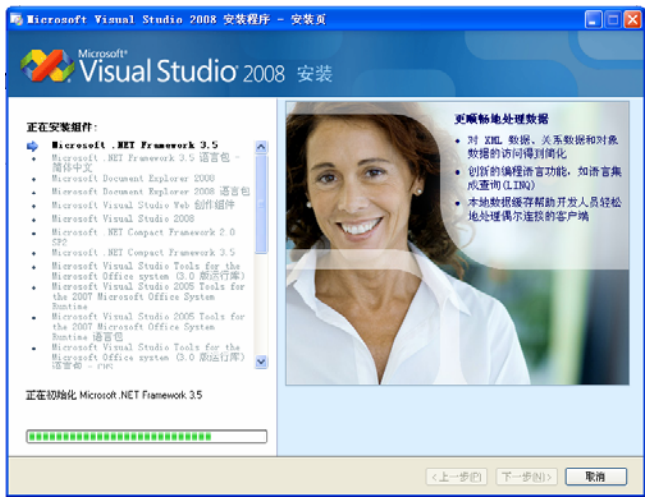


图 C.6　开始安装

（7）安装完毕后向导将返回主界面，此时可以选择继续安装产品文档。

（8）安装完毕后，单击图 C.1 中的“安装产品文档”，把所需要的帮助文档安装到电脑中，方便查阅，如果电脑空间有限，也可以不安装，当查阅的时候，Visual Studio 2008 会自动从网络上搜索你所要查阅的内容。

一切安装完成之后，在开始菜单中可以看到“程序”→“Microsoft Visual Stuido 2008”→“Microsoft Visual Stuido 2008”，单击后即可启动 IDE 开发环境。与早期 Visual Studio 不同，Visual Studio 2008 将所有开发语言都集成在同一个 IDE 开发环境之中，因此不会再有 Visual C++、Visual Basic 这样独立的程序项了。

C.2.2 安装SQL Server 2005

微软 SQL Server 2005 针对不同的目标市场共推出 4 种版本：Express、Workgroup、Standard（标准版）、Enterprise（企业版），不同的版本所提供的可扩展性和性能有所不同，如表 C.1 所示。

表 C.1

功　能	Express	WorkGroup	Standard	Enterprise
CPU 数目	1	2	4	没有限制
内存上限	1 GB	3 GB	操作系统的最大值	内存受限于操作系统支持的上限
数据库大小	4 GB	没有限制	没有限制	没有限制
64 位支持	Windows on Windows(WOW)	WOW	√	√

其中 Express 是微软免费提供的，附带在 Visual Studio 2005 中，安装 Visual Studio 2005 的时候可以选择一起安装。

其他版本的具体安装步骤如下。

（1）关闭所有打开的应用程序，插入 SQL Server 2005 安装光盘，或者从虚拟光驱载入安装镜像文件，光盘自动运行后弹出安装对话框，如图 C.7 所示，在对话框中选择电脑的平台类型。

（2）选择后弹出如图 C.8 所示的开始安装对话框，在此对话框中单击“服务器组件、工具、联机丛书和示例”链接。

图 C.7　平台选择窗口

图 C.8　开始安装界面

（3）接着将出现协议对话框，选择“我接受许可条款和条件”复选框，如图 C.9 所示。

（4）接着将出现检测必备组件的界面，如果检测系统具备安装 SQL Server 2005 的条件，将会显示“已成功安装所需的组件”，如图 C.10 所示，单击“下一步”按钮。

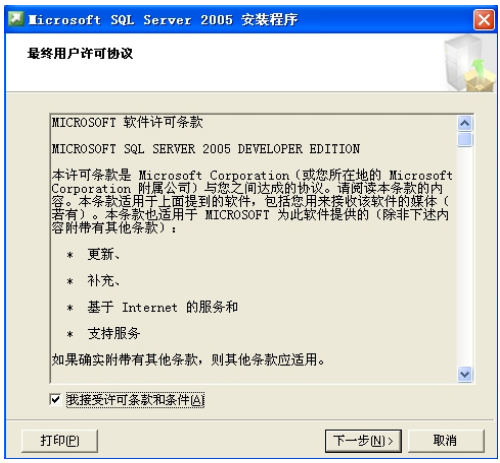


图 C.9　协议界面

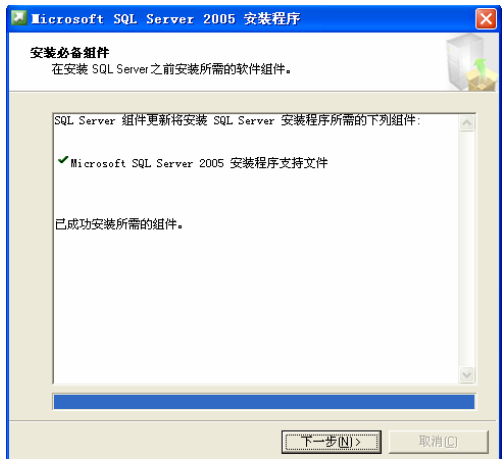


图 C.10　安装必备组件检测界面

（5）在出现“安装向导欢迎界面”对话框中单击“下一步”按钮，弹出“系统配置检查”对话框，如图 C.11 所示。

（6）单击“下一步”按钮弹出“输入注册信息”对话框，输入注册信息后单击“下一步”按钮，弹出“要安装的组件”对话框，第一个选项和最后一个选项必须选择，中间的各种服务，根据需要自主选择，如图 C.12 所示。

图 C.11　系统配置检测对话框

图 C.12　选择安装的组件

（7）单击“下一步”按钮，弹出“实例名”对话框，如图 C.13 所示，可以采用默认的实例，也可以自主给服务实例命名，单击“下一步”按钮。

（8）在弹出的“服务账户”对话框中，如图 C.14 所示，选择“使用内置系统账户”，单击“下一步”按钮。

（9）在弹出的“身份验证模式”对话框中选择“混合模型”，并设置 sa 用户的密码，如图 C.15 所示，单击“下一步”按钮，后面弹出的对话框采用默认设置即可，即单击“下一步”按钮直到安装结束。

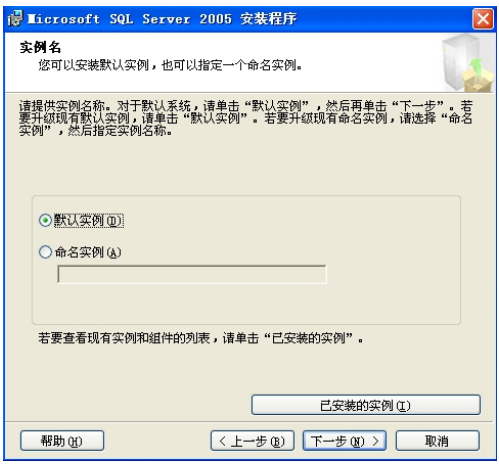


图 C.13　设置实例名

图 C.14　设置服务账户

（10）安装成功后，选择“开始”→“所有程序”→“Microsoft SQL Server 2005”→“SQL Server Management Studio”菜单项，弹出“连接到服务器”对话框，如图 C.16 所示。

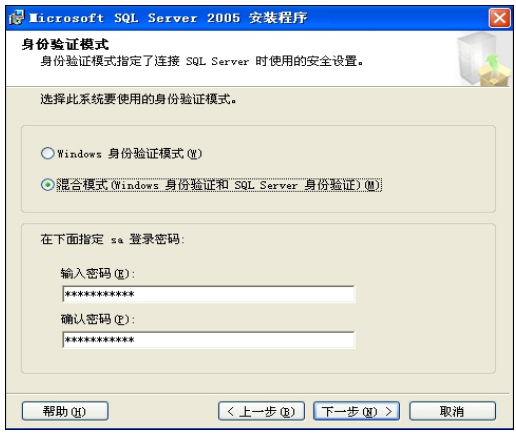


图 C.15　身份认证模式

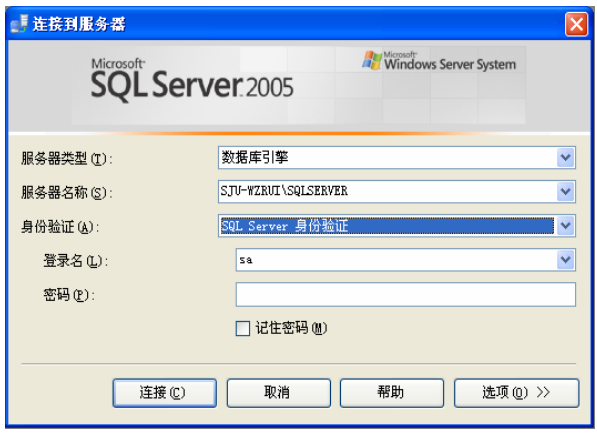


图 C.16　连接服务器

（11）服务器名称选择“本机机器名\实例名”，身份验证选择“SQL Server 身份验证”，登录名输入 sa，密码输入所设置的 sa 的密码，单击“连接”按钮，弹出“Microsoft SQL Server Management Studio”界面，如图 C.17 所示。

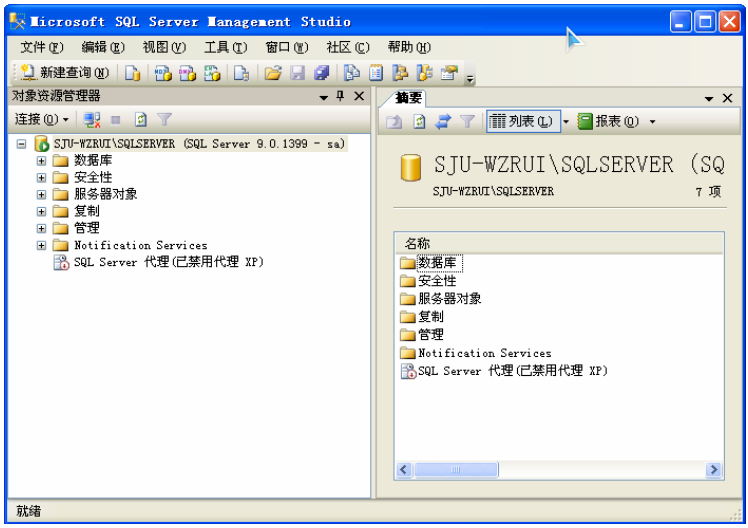


图 C.17　数据库管理器